MIT CSIL • COMPUTER SCIENCE EDUCATION

2026 EDITION

The Missing Semester of Your CS Education

Burmese

ORIGINAL COURSE

Anish Athalye, Jon Gjengset, & Jose Javier Gonzalez Ortiz

Massachusetts Institute of Technology (MIT)

Original Website: https://missing.csail.mit.edu/

MY LOCALIZATION & PUBLICATION

FOSS Myanmar (fossmyanmar.org) & **Vibe Code Tours** (vibecode.tours)

Burmese Website: https://missing-semester-my.github.io/

မာတိကာ (Table of Contents)

သင်တန်း မိတ်ဆက် (Introduction)

ကွန်ပျူတာသိပ္ပံ (CS) သင်တန်းများတွင် OS များ (Operating Systems) မှစ၍ စက်သင်ယူမှု (Machine Learning) အထိ အဆင့်မြင့် ခေါင်းစဉ်များကို သင်ကြားပေးကြသော်လည်း၊ ကျောင်းသားများ ကိုယ်တိုင် ကြိုးစားလေ့လာယူရလေ့ရှိပြီး ပုံမှန် သင်ရိုးများတွင် ထည့်သွင်းသင်ကြားပေးလေ့မရှိသော အရေးကြီးသည့် ဘာသာရပ်တစ်ခု ရှိနေပါသည်—ယင်းမှာ မိမိတို့ အသုံးပြုသည့် tool များကို ကျွမ်းကျင်စွာ ကိုင်တွယ်နိုင်မှုပင် ဖြစ်သည်။ ဤသင်တန်းတွင် Command-line ကို ကျွမ်းကျင်စွာ အသုံးပြုပုံ၊ စွမ်းအားထက်မြက်သော Text Editor များ အသုံးပြုပုံ၊ Version Control စနစ်များ၏ အဆင့်မြင့် လုပ်ဆောင်ချက်များ အသုံးပြုပုံနှင့် အခြား အရေးပါသော အကြောင်းအရာများကို သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။

ကျောင်းသားများသည် မိမိတို့၏ ကျောင်းသားဘဝတွင် ဤ tool များကို အသုံးပြု၍ နာရီပေါင်း ရာချီ (နှင့် သက်မွေးဝမ်းကျောင်း လုပ်ငန်းခွင်တွင် နာရီပေါင်း သောင်းချီ) အချိန်ကုန်လွန်ကြရသဖြင့်၊ ယင်း tool များကို တတ်နိုင်သမျှ ချောမွေ့လွယ်ကူစွာ အသုံးပြုနိုင်အောင် ပြုလုပ်ထားခြင်းသည် အလွန်အကျိုးရှိလှပါသည်။ ဤ tool များကို ကျွမ်းကျင်အောင် သင်ယူခြင်းဖြင့် tool များနှင့် ရုန်းကန်ရသည့် အချိန်များကို လျှော့ချပေးနိုင်ရုံမျှ မက၊ ယခင်က အလွန်ခက်ခဲနက်နဲသည်ဟု ထင်မှတ်ခဲ့သော ပြဿနာများကိုပါ ဖြေရှင်းနိုင်စွမ်း ရရှိစေမည် ဖြစ်ပါသည်။

ယနေ့ခေတ်တွင် AI နည်းပညာသုံး tool များနှင့် လုပ်ငန်းစဉ်များ ပေါ်ထွက်လာခြင်းကြောင့် ဆော့ဖ်ဝဲလ် အင်ဂျင်နီယာ ဘာသာရပ်၏ အသွင်အပြင်များစွာ ပြောင်းလဲလျက် ရှိပါသည်။ ယင်း tool များကို အကန့်အသတ်များနှင့်တကွ သင့်လျော်စွာ အသုံးပြုတတ်ပါက CS ကျွမ်းကျင်သူများအတွက် များစွာ အကျိုး ရှိစေမည် ဖြစ်၍ လက်တွေ့ အသုံးပြုနိုင်သည့် အသိပညာ ရရှိထားရန် လိုအပ်ပါသည်။ AI သည် နယ်ပယ် ပေါင်းစုံတွင် အထောက်အကူပြုသော နည်းပညာဖြစ်သဖြင့် သီးခြား AI သင်ခန်းစာအဖြစ် မထားရှိဘဲ၊ နောက်ဆုံးပေါ် AI tool များနှင့် နည်းလမ်းများကို သက်ဆိုင်ရာ သင်ခန်းစာ တစ်ခုချင်းစီတွင် တိုက်ရိုက် ထည့်သွင်းပေးထားပါသည်။

ဤသင်တန်းကို ဖန်တီးခဲ့သည့် ရည်ရွယ်ချက်ကို ဖတ်ရှုပါ (Motivation) • 📚 အီးဘွတ်ခ် စာအုပ်များ ဒေါင်းလုဒ်ဆွဲရန် (eBook & PDF)။

သင်ရိုးညွှန်းတမ်း (Syllabus)

- **Agentic Coding**
- **Beyond the code (ကုဒ်ရေးသားခြင်းထက် ကျော်လွန်၍)**
- **Code Quality**
- **Command-line ပတ်ဝန်းကျင်**
- **သင်တန်းမိတ်ဆက် + Shell အသုံးပြုနည်းမိတ်ဆက်**
- **Debugging နှင့် Profiling**
- **Development Environment နှင့် Tools များ**
- **2026 Lectures**
- **Packaging and Shipping Code**
- **Version Control နှင့် Git**

လွန်ခဲ့သော နှစ်များမှ အထူး ခေါင်းစဉ်များ (Special topics from previous years)

ကျွန်ုပ်တို့ သင်ကြားပေးသော ခေါင်းစဉ်များသည် တစ်နှစ်နှင့်တစ်နှစ် ကွဲပြားနိုင်ပါသည်။ လွန်ခဲ့သော နှစ်များတွင် သင်ကြားခဲ့သော်လည်း ၂၀၂၆ တွင် မပါဝင်သေးသော ခေါင်းစဉ်များကို လေ့လာလိုသူများအတွက် အောက်ပါအတိုင်း စုစည်းဖော်ပြပေးထားပါသည်။

- *No special topics listed.*

အထွေထွေ အချက်အလက်များ (General information)

သင်ကြားပေးသူများ: ဤသင်တန်းကို Anish၊ Jon နှင့် Jose တို့မှ ပူးတွဲ သင်ကြားပေးပါသည်။
မေးမြန်းရန်: missing-semester@mit.edu သို့ အီးမေးလ်ဖြင့် မေးမြန်းနိုင်ပါသည်။
ဆွေးနွေးပွဲများ: OSSU Discord (မေးခွန်းများအတွက် `#missing-semester-forum` နှင့် တိုက်ရိုက် ဆွေးနွေးရန် `#missing-semester` ကို အသုံးပြုပါ)။

Beyond MIT (MIT အပြင်ဘက် လက်တွေ့ကမ္ဘာ)

Beyond MIT (MIT အပြင်ဘက် လက်တွေ့ကမ္ဘာ) မှ လေ့လာသူများလည်း ဤ အရင်းအမြစ်များမှ အကျိုးကျေးဇူး ရရှိနိုင်ရန်အတွက် မျှဝေပေးထားပါသည်။ အောက်ပါ link များတွင် ဆွေးနွေးချက်များကို ဝင်ရောက် ကြည့်ရှုနိုင်ပါသည်-

- Hacker News (2026, 2020, 2019)
- Lobsters (2026, 2020, 2019)
- r/learnprogramming (2026, 2020, 2019)
- r/programming (2020, 2019)
- X (2026, 2020, 2019)
- Bluesky (2026)
- Mastodon (2026)
- LinkedIn (2026)
- YouTube (2026, 2020, 2019)

ဘာသာပြန်ဆိုချက်များ (Translations)

- Arabic
- Bengali
- Burmese
- Chinese (Simplified)
- Chinese (Traditional, Taiwan)
- German
- Italian
- Japanese
- Kannada
- Korean
- Mongolian
- Persian
- Portuguese
- Russian
- Serbian
- Spanish
- Swedish
- Thai
- Turkish
- Vietnamese

Note: ဤသည်တို့မှာ အသိုင်းအဝိုင်းမှ ပြုလုပ်ထားသော ပြင်ပ ဘာသာပြန် link များ ဖြစ်ကြပြီး၊ ကျွန်ုပ်တို့သည် ၎င်းတို့ကို စစ်ဆေးအတည်ပြုထားခြင်း မရှိပါ။

သင်သည် ဤသင်တန်း၏ သင်ခန်းစာများကို အခြားဘာသာသို့ ဘာသာပြန်ဆိုထားပါက ကျွန်ုပ်တို့၏ စာရင်းတွင် ထည့်သွင်းနိုင်ရန် pull request တင်သွင်းနိုင်ပါသည်။

ကျေးဇူးတင်လွှာ (Acknowledgments)

သင်ခန်းစာ ဗီဒီယိုများ ရိုက်ကူးနိုင်ရန် ကူညီပေးခဲ့ကြသော Elaine Mello နှင့် **MIT Open Learning** တို့ကို ကျေးဇူးတင်ရှိပါသည်။ **SIPB IAP 2026** ၏ အစိတ်အပိုင်းတစ်ခုအဖြစ် ဤသင်တန်းကို ကူညီပံ့ပိုးပေးခဲ့သော Luis Turino / **SIPB** တို့ကိုလည်း အထူး ကျေးဇူးတင်ရှိပါသည်။

Source code.

Source code - MY.

See **here** for contribution & translation guidelines.

ဤသင်တန်းကို သင်ကြားပေးရသည့် ရည်ရွယ်ချက် (Course Motivation)

ပုံမှန် ကွန်ပျူတာသိပ္ပံ ပညာရေးတွင် OS များ (Operating Systems) မှစ၍ ပရိုဂရမ်းမင်း ဘာသာစကားများ နှင့် စက်သင်ယူမှု (Machine Learning) အထိ CS ၏ အဆင့်မြင့် ခေါင်းစဉ်များစွာကို သင်ကြားရလေ့ ရှိပါသည်။ သို့သော် အဖွဲ့အစည်း အများစုတွင် ထည့်သွင်းသင်ကြားပေးလေ့မရှိဘဲ ကျောင်းသားများ ကိုယ်တိုင် လေ့လာယူရန် ချန်ထားလေ့ရှိသော အရေးကြီးသည့် ခေါင်းစဉ်တစ်ခု ရှိနေပါသည်—ယင်းမှာ ကွန်ပျူတာ နည်းပညာ ဂေဟစနစ်နှင့် tool များကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်မှုပင် ဖြစ်သည်။

နှစ်များစွာအတွင်း MIT တွင် သင်တန်းများစွာ ပူးတွဲ သင်ကြားပေးရင်း ကျောင်းသားများစွာသည် မိမိတို့ အသုံးပြုနိုင်သည့် tool များအကြောင်း ဗဟုသုတ နည်းပါးနေသည်ကို မကြာခဏ တွေ့မြင်ခဲ့ရသည်။ ကွန်ပျူတာများကို လက်ဖြင့် ပြုလုပ်ရသည့် အလုပ်များကို အလိုအလျောက် ပြုလုပ်ရန် ဖန်တီးထားခြင်း ဖြစ်သော်လည်း၊ ကျောင်းသားများသည် ထပ်ခါတလဲလဲ လုပ်ဆောင်ရသည့် အလုပ်များကို လက်ဖြင့်ပင် ပြုလုပ်နေကြဆဲ ဖြစ်ပြီး Version Control နှင့် Text Editor ကဲ့သို့သော စွမ်းအားထက်မြက်သည့် tool များကို အပြည့်အဝ အသုံးမချနိုင်ကြပါ။ အကောင်းဆုံး အခြေအနေတွင် အချိန်ကုန်ပြီး အလုပ်မတွင်ဘဲ ဖြစ်ရသကဲ့သို့၊ အဆိုးဆုံး အခြေအနေတွင် ဒေတာ ဆုံးရှုံးခြင်း သို့မဟုတ် အချို့သော အလုပ်များကို မပြီးမြောက်နိုင်ခြင်း အထိ ဖြစ်စေနိုင်ပါသည်။

ဤခေါင်းစဉ်များကို တက္ကသိုလ် သင်ရိုးညွှန်းတမ်းများတွင် ထည့်သွင်းသင်ကြားပေးလေ့မရှိပါ—ကျောင်းသားများအား ဤ tool များကို မည်သို့ အသုံးပြုရမည်၊ သို့မဟုတ် ထိရောက်စွာ အသုံးပြုနည်းကို ပြသပေးခြင်း မရှိသဖြင့် ရိုးရှင်းသင့်သော အလုပ်များတွင် အချိန်နှင့် အားထုတ်မှုများ အဟောသိကံဖွာ ဖြစ်ရပါသည်။ ပုံမှန် CS သင်ရိုးတွင် ကျောင်းသားများ၏ ဘဝကို အလွန် လွယ်ကူ အဆင်ပြေစေနိုင်မည့် ကွန်ပျူတာ နည်းပညာ ဂေဟစနစ်ဆိုင်ရာ အရေးကြီးသော ခေါင်းစဉ်များ လိုအပ်နေပါသည်။

The Missing Semester of Your CS Education

ဤလိုအပ်ချက်ကို ဖြည့်ဆည်းရန်အတွက် ထိရောက်သော ကွန်ပျူတာသိပ္ပံပညာရှင်နှင့် Programmer တစ်ယောက် ဖြစ်လာစေရန် မရှိမဖြစ် လိုအပ်သည်ဟု ကျွန်ုပ်တို့ ယူဆသော ခေါင်းစဉ်များအားလုံး ပါဝင်သည့် သင်တန်းတစ်ခုကို ဖန်တီးခဲ့ပါသည်။ ဤသင်တန်းသည် လက်တွေ့ကျပြီး သင်ကြုံတွေ့ရမည့် အခြေအနေအမျိုးမျိုးတွင် ချက်ချင်း အသုံးချနိုင်မည့် tool များနှင့် နည်းလမ်းများကို လက်တွေ့ မိတ်ဆက်ပေးသွားမည် ဖြစ်ပါသည်။

ဤသင်တန်းတွင် သင်ယူရမည့် အကြောင်းအရာများ၏ လက်တွေ့ စံနမူနာများမှာ-

Command shell

Aliases၊ Scripts နှင့် Build Systems များကို အသုံးပြု၍ ပုံမှန် ထပ်ခါတလဲလဲ ပြုလုပ်ရသော အလုပ်များကို မည်သို့ အလိုအလျောက် (Automate) ပြုလုပ်မည်နည်း။ Text document မှ command များကို ကူးယူ ကူးထည့် (copy-paste) ရသည့် ဒုက္ခများ၊ command ၁၅ ခုကို တစ်ခုပြီးတစ်ခု အစဉ်လိုက် ရိုက်ထည့်ရခြင်းများ၊ argument များ ပို့ရန် မေ့လျော့ခြင်းများ နောက်ထပ် ကြုံတွေ့ရတော့မည် မဟုတ်ပါ။

ဥပမာအားဖြင့်၊ Shell History အတွင်း မြန်ဆန်စွာ ရှာဖွေနိုင်ခြင်းသည် အချိန်များစွာ အသက်သာစေပါသည်။ အောက်ပါ စံနမူနာတွင် `convert` command များအတွက် shell history ကို ရှာဖွေအသုံးပြုပုံ အကွက်ဆန်း အချို့ကို ဖော်ပြထားပါသည်-

📹 **Video Demo (History)**

Scan QR code or visit link to watch demo:

https://missing-semester-my.github.io/static/media/demos/history.mp4

Version control (Git)

Version control ကို စနစ်တကျ မှန်ကန်စွာ အသုံးပြုနည်း၊ ယင်းကို အသုံးချ၍ မတော်တဆ ပျက်စီးမှုများမှ ကာကွယ်နည်း၊ အခြားသူများနှင့် ပူးပေါင်းဆောင်ရွက်နည်း၊ နှင့် ဒုက္ခပေးနေသော အပြောင်းအလဲများကို လျင်မြန်စွာ ရှာဖွေ ခွဲထုတ်နည်း။ `rm -rf; git clone` ပြုလုပ်စရာ မလိုတော့ပါ။ Merge conflict များလည်း (အနည်းဆုံးတော့ ပိုမို နည်းပါးသွားမည်) ဖြစ်ပါသည်။ Comment ပိတ်ထားသော စာကြောင်းကြီးများ ချန်ထားစရာ မလိုတော့ပါ။ ကုဒ် ပျက်စီးသွားသည့် နေရာကို ရှာဖွေရန် စိုးရိမ်စရာ မလိုတော့ပါ။ ဤသင်တန်း

တွင် pull request များ အသုံးပြု၍ အခြားသူများ၏ project များတွင် မည်သို့ ပါဝင်ကူညီရမည်ကိုပါ သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။

အောက်ပါ ဥပမာတွင် `git bisect` ကို အသုံးပြု၍ Unit Test ကို ပျက်စီးစေခဲ့သော commit ကို ရှာဖွေပြီး `git revert` ဖြင့် ပြန်လည် ပြင်ဆင်ပုံကို ဖော်ပြထားပါသည်-

📹 **Video Demo (Git)**

Scan QR code or visit link to watch demo:

https://missing-semester-my.github.io/static/media/demos/git.mp4

Text editing (Vim)

Local နှင့် Remote စက်များရှိ file များကို command-line မှနေ၍ ထိရောက်စွာ Edit ပြုလုပ်နည်းနှင့် အဆင့်မြင့် Editor များ၏ လုပ်ဆောင်ချက်များကို အသုံးချနည်း။ File များကို ရှေ့နောက် ကူးယူနေစရာ မလိုတော့ပါ။ ထပ်ခါတလဲလဲ file ပြင်ဆင်နေစရာ မလိုတော့ပါ။

Vim macros များသည် Vim ၏ အကောင်းဆုံး လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပြီး အောက်ပါ ဥပမာတွင် HTML table တစ်ခုကို CSV format သို့ Vim macro အသုံးပြု၍ လျင်မြန်စွာ ပြောင်းလဲပုံကို ဖော်ပြထားပါသည်-

📹 **Video Demo (Vim)**

Scan QR code or visit link to watch demo:

https://missing-semester-my.github.io/static/media/demos/vim.mp4

Remote machines (SSH & Terminal Multiplexing)

SSH keys များနှင့် terminal multiplexing များကို အသုံးပြု၍ remote machine များနှင့် အလုပ်လုပ်ရာတွင် စနစ်တကျ အဆင်ပြေစေရန် ပြုလုပ်နည်း။ Command နှစ်ခုကို ပြိုင်တူ လွှတ်ရန်အတွက် Terminal တွေ အများကြီး ဖွင့်ထားစရာ မလိုတော့ပါ။ ချိတ်ဆက်တိုင်း password ပြန်ရိုက်နေစရာ မလိုတော့ပါ။ အင်တာနက် လိုင်းကျသွားခြင်း သို့မဟုတ် Laptop reboot လုပ်လိုက်ရခြင်းကြောင့် အရာအားလုံး ဆုံးရှုံးသွားရခြင်းမျိုး မရှိတော့ပါ။

အောက်ပါ ဥပမာတွင် `tmux` ကို အသုံးပြု၍ remote server များပေါ်တွင် session များကို အသက်ရှင်လျက် ထိန်းသိမ်းထားပုံနှင့် `mosh` အသုံးပြု၍ network ပြောင်းလဲမှုကို ထိန်းသိမ်းပေးပုံကို ဖော်ပြထားပါသည်-

📹 **Video Demo (Ssh)**

Scan QR code or visit link to watch demo:

https://missing-semester-my.github.io/static/media/demos/ssh.mp4

Finding files (ဖိုင်များ ရှာဖွေခြင်း)

သင် ရှာဖွေနေသော file များကို လျင်မြန်စွာ ရှာဖွေနည်း။ မိမိ လိုချင်သော ကုဒ် ပါသည့် file တွေ့သည့်အထိ project ထဲက file တစ်ခုချင်းစီလိုက် နှိပ်ကြည့်နေစရာ မလိုတော့ပါ။

အောက်ပါ ဥပမာတွင် `fd` ဖြင့် file များကို မြန်ဆန်စွာ ရှာဖွေပြီး `rg` ဖြင့် ကုဒ် အစိတ်အပိုင်းများကို ရှာဖွေပုံ၊ ထို့ပြင် `fasd` ဖြင့် မကြာသေးမီက အသုံးပြုခဲ့သော file/folder သို့ လျင်မြန်စွာ `cd` နှင့် `vim` ဝင်ရောက်ပုံတို့ကို ဖော်ပြထားပါသည်-

📹 **Video Demo (Find)**

Scan QR code or visit link to watch demo:

https://missing-semester-my.github.io/static/media/demos/find.mp4

ဒေတာ ပြုပြင်စီမံခြင်း (Data Wrangling)

Command-line မှ တိုက်ရိုက် ဒေတာနှင့် ဖိုင်များကို လျင်မြန် လွယ်ကူစွာ ပြုပြင်ခြင်း၊ ကြည့်ရှုခြင်း၊ Parse လုပ်ခြင်း၊ Plot ဆွဲခြင်းနှင့် တွက်ချက်ခြင်းများ ပြုလုပ်နည်း။ Log file များထဲမှ ကူးယူ ကူးထည့် စရာ မလိုတော့ပါ။ ဒေတာ စာရင်းအင်းများကို လက်ဖြင့် တွက်ချက်နေစရာ မလိုတော့ပါ။

Code quality and continuous integration

Autoformatting၊ Linting၊ Testing နှင့် Code coverage tool များကို အသုံးပြု၍ ကုဒ် အရည်အသွေး တိုးတက်အောင် ပြုလုပ်နည်း။ ရုပ်ဆိုးသော ကုဒ်များ၊ မလိုလားအပ်သော အမှားဟောင်းများ ပြန်ဖြစ်ခြင်းများ၊ မိမိစက်တွင် အလုပ်လုပ်ပြီး အခြားသူ စက်တွင် ပျက်စီးသွားသော ကုဒ်များ မရှိတော့ပါ။

Beyond the code (ကုဒ်ရေးသားခြင်းထက် ကျော်လွန်၍)

ကောင်းမွန်သော Documentation ရေးသားနည်း၊ Open-source ထိန်းသိမ်းသူများနှင့် ရှင်းလင်းစွာ ဆက်သွယ်နည်း၊ တိကျသော Issue များ တင်သွင်းနည်းနှင့် Merge လုပ်ခံရမည့် Pull Request များ ပါဝင်ကူညီနည်း။

နိဂုံး (Conclusion)

ဤအကြောင်းအရာများနှင့် အခြား အကြောင်းအရာများစွာကို သင်ခန်းစာများအတွင်း သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။ ဇန်နဝါရီ မတိုင်မီ စောစီးစွာ လေ့လာလိုပါက ယခင် သင်တန်း **2020 သင်ခန်းစာများ** ကိုလည်း ဝင်ရောက် ကြည့်ရှုနိုင်ပါသည်။

Happy hacking,

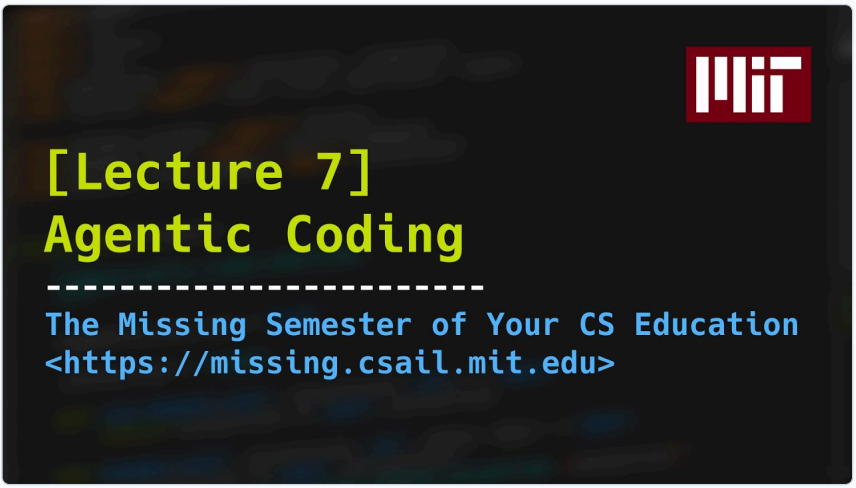
Anish၊ **Jon** နှင့် **Jose**

Agentic Coding

အနှစ်ချုပ် - ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်ရေး လုပ်ငန်းစဉ်များအတွက် AI coding agent များကို ထိရောက်စွာ အသုံးပြုနည်းကို လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Agentic Coding



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=sTdz6PZoAnw

Coding agent များသည် ဖိုင်များ ဖတ်ခြင်း/ရေးခြင်း၊ ဝက်ဘ်ရှာဖွေခြင်းနှင့် shell command များကို စေခိုင်းခြင်း စသည့် tool များကို ရယူသုံးစွဲနိုင်သော အပြန်အလှန်ပြောဆိုနိုင်သည့် (conversational) AI model များဖြစ်ကြသည်။ ၎င်းတို့ကို IDE ထဲတွင်သော်လည်းကောင်း၊ သီးခြား command-line သို့မဟုတ် GUI tool များအဖြစ်သော်လည်းကောင်း အသုံးပြုနိုင်သည်။ Coding agent များသည် အလွန် ကိုယ်ပိုင်ဆုံးဖြတ်လုပ်ဆောင်နိုင်စွမ်း (autonomous) ရှိပြီး စွမ်းဆောင်ရည် ထက်မြက်သော tool များ ဖြစ်ကြသဖြင့် လုပ်ဆောင်ချက်မျိုးစုံတွင် အသုံးပြုနိုင်စေသည်။

ဤသင်ခန်းစာသည် **Development Environment and Tools** (https://missing-semester-my.github.io/2026/development-environment/) သင်ခန်းစာမှ AI ကိုအသုံးပြုသည့် ဖွံ့ဖြိုးတိုးတက်ရေးဆိုင်ရာ အကြောင်းအရာများပေါ်တွင် အခြေခံထားပါသည်။ အမြန် demo အနေဖြင့် **AI-powered development** (https://missing-semester-my.github.io/2026/development-environment/#ai-powered-development) အပိုင်းမှ ဥပမာကို ဆက်လက် လေ့လာကြည့်ကြပါစို့-

```
from urllib.request import urlopen

def download_contents(url: str) -> str:
    with urlopen(url) as response:
        return response.read().decode('utf-8')

def extract(content: str) -> list[str]:
    import re
    pattern = r'\[.*?\]\((.*?)\)'
    return re.findall(pattern, content)

print(extract(download_contents("https://raw.githubusercontent.com/missing-seme
ster/missing-semester/refs/heads/master/_2026/development-environment.md")))
```

ကျွန်ုပ်တို့သည် coding agent တစ်ခုအား အောက်ပါ task ကို prompt ပေး၍ စမ်းသပ်ကြည့်နိုင်သည်-

```
Turn this into a proper command-line program, with argparse for argument parsin
g. Add type annotations, and make sure the program passes type checking.
```

Agent သည် ဖိုင်ကို နားလည်အောင် ပထမဦးစွာ ဖတ်ရှုမည်ဖြစ်ပြီး၊ ထို့နောက် ပြင်ဆင်မှုများ ပြုလုပ်ကာ နောက်ဆုံးတွင် type annotation များ မှန်ကန်ကြောင်း သေချာစေရန် type checker ကို စေခိုင်း (invoke) မည်ဖြစ်သည်။ အကယ်၍ type checking ကျရှုံးသည်အထိ အမှားပြုလုပ်မိပါက ၎င်းသည် အကြိမ်ကြိမ် ပြန်လည်ပြင်ဆင် (iterate) ပါလိမ့်မည်။ သို့သော် ဤသည်မှာ ရိုးရှင်းသော task တစ်ခုဖြစ်သဖြင့် ထိုသို့ဖြစ်နိုင်ခြေနည်းပါးသည်။ Coding agent များသည် အန္တရာယ်ရှိနိုင်သော tool များကို ရယူသုံးစွဲနိုင်သောကြောင့် မူလအစ

အတိုင်း (by default) agent harness များသည် tool ခေါ်ယူမှုများကို အတည်ပြုရန် အသုံးပြုသူထံ အကြောင်းကြားမေးမြန်းလေ့ရှိသည်။

> အကယ်၍ coding agent သည် အမှားပြုလုပ်မိပါက — ဥပမာ သင့်ထံတွင် `mypy` binary ကို `$PATH` တွင် တိုက်ရိုက် ရရှိနိုင်သော်လည်း agent က `python -m mypy` ကို ခေါ်ယူရန် ကြိုးပမ်းနေပါက — ၎င်းအား လမ်းကြောင်းမှန်သို့ ပြန်လည်ရောက်ရှိစေရန် စာသားဖြင့် feedback ပေးနိုင်သည်။

Coding agent များသည် multi-turn တို့ပြန်ဆက်ဆံမှုကို ထောက်ပံ့ပေးသဖြင့် agent နှင့် အပြန်အလှန် စကားပြောဆိုရင်း အလုပ်ကို ထပ်ခါတလဲလဲ ပြင်ဆင်သွားနိုင်သည်။ Agent သည် လမ်းကြောင်းမှားသို့ သွားနေပါက ကြားဖြတ်တားဆီး (interrupt) ပင် ပြုလုပ်နိုင်သည်။ သင့်တော်သော စဉ်းစားပုံစံ (mental model) တစ်ခုမှာ အလုပ်သင် (intern) တစ်ဦး၏ မန်နေဂျာကဲ့သို့ စဉ်းစားခြင်း ဖြစ်သည်- အလုပ်သင်သည် သေးငယ်စွာပင်သော အလုပ်များကို လုပ်ဆောင်ပေးမည်ဖြစ်သော်လည်း လမ်းညွှန်မှု လိုအပ်မည်ဖြစ်ပြီး၊ တစ်ခါတစ်ရံတွင် အမှားများ ပြုလုပ်မိသဖြင့် ပြင်ဆင်ပေးရန် လိုအပ်မည်ဖြစ်သည်။

> ပိုမို ရှင်းလင်းသော demo တစ်ခုအတွက်၊ ထွက်ပေါ်လာသည့် script ကို run ကြည့်ရန် (run) agent ထံ ထပ်မံ မေးမြန်းကြည့်ပါ။ Output များကို လေ့လာပြီး ပြောင်းလဲမှုတစ်ခု ပြုလုပ်ပေးရန် တောင်းဆို ကြည့်ပါ (ဥပမာ absolute URL များကိုသာ ထည့်သွင်းပေးရန် တောင်းဆိုပါ)။

AI model များနှင့် agent များ မည်သို့ အလုပ်လုပ်သနည်း

ခေတ်မီ **large language models (LLMs)** (https://en.wikipedia.org/wiki/Large_language_model) များနှင့် agent harness များကဲ့သို့သော အခြေခံအဆောက်အအုံများ၏ အတွင်းပိုင်း အလုပ်လုပ်ပုံကို အသေးစိတ် ရှင်းလင်းပြသခြင်းသည် ဤသင်ခန်းစာ၏ နယ်ပယ်ထက် ကျော်လွန်နေပါသည်။ သို့သော်လည်း အဓိက အယူအဆ အချို့ကို ခြုံငုံနားလည်ထားခြင်းသည် ဤရှေ့တန်းရောက် နည်းပညာ (bleeding edge technology) ကို ထိရောက်စွာ *အသုံးပြု*ရန် နှင့် ယင်း၏ ကန့်သတ်ချက်များကို နားလည်ရန်အတွက် အထောက်အကူဖြစ်စေသည်။

LLM များကို prompt စာသားများ (inputs) ပေးထားသည့်အခါ completion စာသားများ (outputs) ၏ ဖြစ်တန်းခြေ ဖြန့်ကျက်မှု (probability distribution) ကို ပုံဖော်ပေးခြင်းအဖြစ် ရှုမြင်နိုင်သည်။ LLM inference (ဥပမာ စကားပြောဆိုနိုင်သော chat app တစ်ခုသို့ မေးခွန်းတစ်ခု ပေးလိုက်သည့်အခါ ဖြစ်ပျက်လာသည့် အရာ) သည် ဤ probability distribution မှ *sample* ရယူခြင်း ဖြစ်သည်။ LLM များတွင် သတ်မှတ်ထားသော *context window* ရှိပြီး၊ ၎င်းသည် input နှင့် output စာသားများ၏ အများဆုံး အရှည်ပမာဏ ဖြစ်သည်။

Conversational chat နှင့် coding agent များကဲ့သို့သော AI tool များကို ဤအခြေခံအယူအဆပေါ်တွင် တည်ဆောက်ထားခြင်းဖြစ်သည်။ Multi-turn တို့ပြန်ဆက်ဆံမှုများအတွက်၊ chat app များနှင့် agent များသည် turn marker များကို အသုံးပြုပြီး၊ အသုံးပြုသူထံမှ prompt အသစ်တစ်ခု ရရှိတိုင်း စကားပြောဆိုမှု သမိုင်းကြောင်း တစ်ခုလုံးကို prompt စာသားအဖြစ် ပေးပို့ကာ၊ အသုံးပြုသူ prompt တစ်ခုလျှင် LLM inference တစ်ကြိမ် စေခိုင်းသည်။ Tool-calling agent များအတွက်မူ harness သည် အချို့သော LLM output များကို tool တစ်ခုအား စေခိုင်းရန် တောင်းဆိုမှုအဖြစ် အဓိပ္ပာယ်ဖော်ယူပြီး၊ harness က tool ခေါ်ယူမှု၏ ရလဒ်များကို prompt စာသား၏ အစိတ်အပိုင်းအဖြစ် model ထံ ပြန်လည်ပေးပို့သည် (ထို့ကြောင့် tool call/response တစ်ခုစီအတွက် LLM inference ပြန်လည် run သည်)။ Tool-calling agent များ၏ အဓိက အယူအဆများကို **code စာကြောင်းရေ ၂၀၀ ဖြင့် ရေးသားအကောင်အထည်ဖော်နိုင်သည်** (https://www.mihaileric.com/The-Emperor-Has-No-Clothes/)။

သီးသန့်လုံခြုံမှု (Privacy)

AI coding tool အများစုသည် ၎င်းတို့၏ ပုံမှန် configuration တွင် သင့်ဒေတာ အများအပြားကို cloud ထံ ပေးပို့ကြသည်။ အချို့အခါများတွင် harness သည် Local တွင် run ပြီး LLM inference က cloud တွင် run သော်လည်း၊ အချို့အခါများတွင် ဆော့ဖ်ဝဲလ်၏ ပိုမိုများပြားသော အစိတ်အပိုင်းများက cloud တွင် run နေ

လေ့ရှိသည် (ဥပမာ ဝန်ဆောင်မှုပေးသူသည် သင့် repository တစ်ခုလုံး၏ မိတ္တူနှင့် AI tool နှင့် ပြုလုပ်ခဲ့သမျှ တုံ့ပြန်ဆောင်ရွက်မှု အားလုံးကို ရရှိသွားနိုင်ပါသည်)။

တော်တော်လေး ကောင်းမွန်သည့် open-source AI coding tool များနှင့် open-source LLM များ ရှိကြသော်လည်း (proprietary model များလောက်တော့ ကောင်းမွန်ခြင်း မရှိသေးပါ)၊ လက်ရှိအချိန်တွင် အသုံးပြုသူ အများစုအတွက် ရှေ့တန်းရောက် open LLM များကို Local တွင် run ရန်မှာ ဟာဒ်ဝဲလ် ကန့်သတ်ချက်များကြောင့် ဖြစ်နိုင်ခြေ မရှိသေးပါ။

အသုံးပြုနိုင်သည့် အခြေအနေများ (Use cases)

Coding agent များကို လုပ်ဆောင်ချက် အမျိုးအစားများစွာအတွက် အသုံးဝင်စွာ အသုံးပြုနိုင်ပါသည်။ ဥပမာအချို့မှာ-

- **Feature အသစ်များကို အကောင်အထည်ဖော်ခြင်း။** အထက်ပါ ဥပမာအတိုင်း coding agent ထံ feature တစ်ခုကို အကောင်အထည်ဖော်ပေးရန် တောင်းဆိုနိုင်သည်။ ကောင်းမွန်သော specification ပေးပို့ခြင်းသည် နည်းပညာထက် ပန်းချီပညာကဲ့သို့ ပိုမိုဆန်းကြယ်သည်- သင့်ထံမှ agent သို့ ပေးသည့် input သည် သင်ဖြစ်စေချင်သည်များကို လုပ်ဆောင်နိုင်လောက်အောင် (အနည်းဆုံးတော့ သင် ထပ်မံပြင်ဆင်နိုင်ရန် လမ်းကြောင်းမှန်သို့ ဦးတည်စေရန်) ရှင်းလင်းပြည့်စုံရမည်ဖြစ်သော်လည်း၊ သင့်ကိုယ်တိုင် အလုပ်များလွန်းသွားသည်အထိ အသေးစိတ် လွန်မနေသင့်ပေ။ Test-driven development သည် အထူး ထိရောက်နိုင်သည်- စမ်းသပ်ချက်များ (tests) ရေးသားပါ (သို့မဟုတ် စမ်းသပ်ချက်များ ရေးသားရာတွင် ကူညီရန် coding agent ကို အသုံးပြုပါ)၊ ၎င်းတို့သည် သင်လိုချင်သောအရာများကို မိမိရရ ပုံဖော်ပေးနိုင်ကြောင်း စစ်ဆေးပါ၊ ထို့နောက် ၎င်း feature ကို အကောင်အထည်ဖော်ပေးရန် coding agent ကို တောင်းဆိုပါ။ Model များသည် စဉ်ဆက်မပြတ် တိုးတက်လျက်ရှိရာ model များ မည်မျှ စွမ်းဆောင်နိုင်သည်ဆိုသည်ကို သင့်အနေဖြင့် အမြဲမပြတ် လေ့လာစမ်းသပ်နေရန် လိုအပ်မည်ဖြစ်သည်။

```
> ကျွန်ုပ်တို့သည် ဤ Tufte-style sidenote များကို <a href="https://github.com/missing-semester/missing-semester/pull/345" class="ext-link">အကောင်အထည်ဖော်ရန်</a> <span class="ext-url-print">(https://github.com/missing-semester/missing-semester/pull/345)</span> Claude Code ကို အသုံးပြုခဲ့ပါသည်။
```

No need to demo this, since the intro of a lecture was a small demo of adding a new feature.

- **အမှားများကို ပြင်ဆင်ခြင်း။** သင့်ထံတွင် compiler၊ linter၊ type checker သို့မဟုတ် test များမှ error များ ရှိနေပါက agent ထံ ၎င်းတို့ကို ပြင်ဆင်ပေးရန် တောင်းဆိုနိုင်သည်၊ ဥပမာ “fix the issues with mypy” ကဲ့သို့သော prompt မျိုး ပေးနိုင်သည်။ Coding model များကို feedback loop ထဲသို့ ရောက်ရှိစေနိုင်လျှင် အထူးပင် ထိရောက်မှုရှိရာ model အနေဖြင့် ကျရှုံးနေသော check ကို တိုက်ရိုက် run နိုင်အောင် ပြင်ဆင်ထားပါ၊ ထိုအခါ model သည် ကိုယ်တိုင် ပြန်လည်ပြင်ဆင်နိုင်မည်ဖြစ်သည်။ အကယ်၍ ဤသို့ ပြုလုပ်ရန် အဆင်မပြေပါက model သို့ feedback ကို ကိုယ်တိုင် ပေးပို့နိုင်သည်။

```
> missing-semester repo ၏ commit <a href="https://github.com/missing-semester/
missing-semester/commit/f552b5523462b22b8893a8404d2110c4e59613dd" class="ext-li
nk">f552b55</a> <span class="ext-url-print">(https://github.com/missing-semeste
r/missing-semester/commit/f552b5523462b22b8893a8404d2110c4e59613dd)</span> တွင်
ကျွန်ုပ်တို့သည် Claude Code အား "Review the agentic coding lecture for typos and gra
mmatical issues" ဟု prompt ပေးခဲ့ပြီး၊ တွေ့ရှိရသော ပြဿနာများကို ပြင်ဆင်ရန် ထပ်မံ ခိုင်းစေခဲ့ရာ၊
၎င်းတို့ကို commit <a href="https://github.com/missing-semester/missing-semester/com
mit/f1e1c417adba6b4149f7eef91ff5624de40dc637" class="ext-link">f1e1c41</a> <spa
n class="ext-url-print">(https://github.com/missing-semester/missing-semester/c
ommit/f1e1c417adba6b4149f7eef91ff5624de40dc637)</span> တွင် commit ပြုလုပ်ခဲ့သည်။
```

Demo a coding agent fixing the bug in https://github.com/anishathalye/dotbot/commit/cef40c902ef0f52f4841534131 42b5154bbc5e99.

Write the failing tests to demo the bug, and then ask the agent to fix. Prepped in branch demo-bugfix.

Can run the failing test with:

```
hatch test tests/test_cli.py::test_issue_357
```

Can prompt coding agent with:

```
There is a bug I wrote a failing test for, you can repro it with `hatch test te
sts/test_cli.py::test_issue_357`. Fix the bug.
```

Get it to commit the changes.

- **Refactoring ပြုလုပ်ခြင်း။** Coding agent များကို အသုံးပြု၍ ကုဒ်များကို နည်းလမ်းမျိုးစုံဖြင့် refactor ပြုလုပ်နိုင်သည်၊ ဥပမာ method တစ်ခု၏ အမည် ပြောင်းလဲခြင်းကဲ့သို့ ရိုးရှင်းသော task များမှသည် (ဤကဲ့သို့သော refactoring မျိုးကို **code intelligence** (https://missing-semester-my.github.io/2026/development-environment/#code-intelligence-and-language-servers) ကလည်း ထောက်ပံ့ပေးသည်) လုပ်ဆောင်ချက်တစ်ခုကို သီးခြား module တစ်ခုအဖြစ် ခွဲထုတ်ခြင်းကဲ့သို့ ရှုပ်ထွေးသော task များအထိ ပြုလုပ်နိုင်သည်။

```
> ကျွန်ုပ်တို့သည် agentic coding ကို သီးခြား သင်ခန်းစာတစ်ခုအဖြစ် <a href="https://github.c
om/missing-semester/missing-semester/pull/344" class="ext-link">ခွဲထုတ်ရန်</a> <sp
an class="ext-url-print">(https://github.com/missing-semester/missing-semester/
pull/344)</span> Claude Code ကို အသုံးပြုခဲ့ပါသည်။
```

Show usage in Missing Semester, point out that the agent did make some mistakes.

- **Code review ပြုလုပ်ခြင်း။** Coding agent များကို ကုဒ်များ စစ်ဆေးပေးရန် (review) တောင်းဆို နိုင်သည်။ “review my latest changes that are not yet committed” ကဲ့သို့သော အခြေခံ လမ်းညွှန်ချက် များ ပေးနိုင်သည်။ အကယ်၍ သင်သည် pull request တစ်ခုကို review ပြုလုပ်လိုပြီး သင့် coding agent က web fetch ကို ထောက်ပံ့ပါက သို့မဟုတ် သင့်ထံတွင် **GitHub CLI** (https://cli.github.com/) ကဲ့သို့သော command-line tool များ ထည့်သွင်းထားပါက coding agent ထံ “Review the pull request {link}” ဟု ပင် တောင်းဆိုနိုင်ပြီး ၎င်းက ကျန်သည်များကို လုပ်ဆောင်ပေးသွားမည်ဖြစ်သည်။

In Porcupine repo, prompt agent with:

```
Review this PR: https://github.com/anishathalye/porcupine/pull/39
```

- **Codebase ကို နားလည်သဘောပေါက်ခြင်း။** Coding agent ထံ codebase နှင့် ပတ်သက်သည့် မေးခွန်း များ မေးမြန်းနိုင်ပြီး၊ ဤသည်မှာ အဖွဲ့သစ် သို့မဟုတ် project သစ်သို့ စတင်ဝင်ရောက်သူများ (onboarding) အတွက် အထူးပင် အသုံးဝင်ပါသည်။

Some prompts to try in the missing-semester repo:

```
How do I run this site locally?

How are the social preview cards implemented?
```

- **Shell အဖြစ် အသုံးပြုခြင်း။** Coding agent အား task တစ်ခုကို ဖြေရှင်းရန် သီးခြား tool တစ်ခုကို အသုံးပြုခိုင်းနိုင်သည်၊ ထို့ကြောင့် “use the find command to find all files older than 30 days” သို့မဟုတ် “use mogrify to resize all the jpgs to 50% of their original size” ကဲ့သို့ လူစကား (natural language) အသုံးပြု၍ shell command တစ်ခုကို စေခိုင်းနိုင်သည်။

In Dotbot repo, prompt agent with:

```
Use the ag command to find all Python renaming imports
```

- **Vibe coding။** Agent များသည် သင့်ကိုယ်တိုင် code စာကြောင်းတစ်ကြောင်းမျှ ရေးသားရန် မလိုဘဲ အချို့သော application များကို အကောင်အထည်ဖော်နိုင်သည်အထိ စွမ်းဆောင်ရည် ထက်မြက်ကြ သည်။

```
> နည်းပြတစ်ဦးမှ vibe-code ပြုလုပ်ခဲ့သည့် လက်တွေ့ကမ္ဘာ project ၏ <a href="https://github.com/cleanlab/office-presence-dashboard" class="ext-link">ဥပမာတစ်ခုကို ဤနေရာတွင် ကြည့်နိုင်ပါသည်</a> <span class="ext-url-print">(https://github.com/cleanlab/office-presence-dashboard)</span>။
```

In missing-semester repo, prompt agent with:

```
Make this site look retro.
```

အဆင့်မြင့် Agent များ (Advanced agents)

ဤနေရာတွင် ကျွန်ုပ်တို့သည် coding agent များကို ပိုမို အဆင့်မြင့်စွာ အသုံးပြုသည့် ပုံစံများနှင့် စွမ်းဆောင်ရည်များကို အကျဉ်းချုပ် ဖော်ပြပေးသွားမည်ဖြစ်သည်။

- **ပြန်လည်အသုံးပြုနိုင်သော prompt များ။** ပြန်လည်အသုံးပြုနိုင်သော prompt သို့မဟုတ် template များကို ဖန်တီးပါ။ ဥပမာအားဖြင့် သီးခြားနည်းလမ်းဖြင့် code review ပြုလုပ်ရန် အသေးစိတ် prompt တစ်ခုကို ရေးသားပြီး ပြန်လည်အသုံးပြုနိုင်သော prompt အဖြစ် သိမ်းဆည်းထားနိုင်သည်။

```
> Agent tool များသည် လျှင်မြန်စွာ တိုးတက်ပြောင်းလဲလျက်ရှိသည်။ အချို့သော tool များတွင် သီးခြား feature အဖြစ် Reusable prompt များကို မသုံးတော့ပါ (deprecated)။ ဥပမာ Codex နှင့် Claude Code တို့တွင် ၎င်းတို့ကို <a href="https://code.claude.com/docs/en/skills" class="ext-link">skills</a> <span class="ext-url-print">(https://code.claude.com/docs/en/skills)</span> များဖြင့် <a href="https://developers.openai.com/codex/custom-prompts" class="ext-link">အစားထိုးလိုက်ပါပြီ</a> <span class="ext-url-print">(https://developers.openai.com/codex/custom-prompts)</span>။
```

- **ပြိုင်တူခိုင်းနိုင်သော agent များ (Parallel agents)။** Coding agent များသည် နှေးကွေးနိုင်ပါသည်- သင် prompt ပေးလိုက်ပါက ပြဿနာတစ်ခုအတွက် မိနစ်ပေါင်းများစွာ အလုပ်လုပ်နေနိုင်သည်။ သင်သည် agent မိတ္တူအများအပြားကို တစ်ပြိုင်နက်တည်း run ထားနိုင်ပြီး၊ ၎င်းတို့အား အလုပ်တစ်ခုတည်းကို ခိုင်းစေခြင်း (LLM များသည် stochastic ဖြစ်သဖြင့် တစ်ခုတည်းသော အရာကို အကြိမ်ကြိမ် run ၍ အကောင်းဆုံး အဖြေကို ရယူခြင်းက အသုံးဝင်နိုင်သည်) သို့မဟုတ် ကွဲပြားသော အလုပ်များကို ခိုင်းစေခြင်း (ဥပမာ တစ်ခုနှင့်တစ်ခု မထပ်သော feature နှစ်ခုကို တစ်ပြိုင်နက်တည်း အကောင်အထည်ဖော်ခြင်း) ပြုလုပ်နိုင်သည်။ ကွဲပြားသော agent ၏ ပြောင်းလဲမှုများ အချင်းချင်း နှောင့်ယှက်မှု မရှိစေရန် **version control** (https://missing-semester-my.github.io/2026/version-control/) သင်ခန်းစာတွင် ဆွေးနွေးခဲ့သည့် **git worktrees** (https://git-scm.com/docs/git-worktree) များကို အသုံးပြုနိုင်သည်။
- **MCP များ။** *Model Context Protocol* ကို ကိုယ်စားပြုသော MCP သည် သင့် coding agent များကို tool များနှင့် ချိတ်ဆက်ရန် အသုံးပြုနိုင်သည့် open protocol တစ်ခု ဖြစ်သည်။ ဥပမာ ဤ **Notion MCP server** (https://github.com/makenotion/notion-mcp-server) သည် သင့် agent အား Notion document များကို ဖတ်/ရေး ပြုလုပ်စေနိုင်ပြီး၊ “{Notion doc} တွင် ချိတ်ဆက်ထားသော spec ကို ဖတ်ပါ၊ အကောင်အထည်ဖော်မှု အစီအစဉ်ကို Notion စာမျက်နှာသစ်အဖြစ် မူကြမ်းရေးပါ၊ ထို့နောက် စာမူကြမ်း (prototype) ကို အကောင်အထည်ဖော်ပါ” ကဲ့သို့သော အခြေအနေများတွင် အသုံးပြုနိုင်စေသည်။ MCP များကို ရှာဖွေရန် **Pulse** (https://www.pulsemcp.com/servers) နှင့် **Glama** (https://glama.ai/mcp/servers) ကဲ့သို့သော directory များကို အသုံးပြုနိုင်သည်။

- **Context စီမံခန့်ခွဲမှု။** ကျွန်ုပ်တို့ အထက်တွင် ဖော်ပြခဲ့သည့်အတိုင်း coding agent များကို အခြေခံထားသော LLM များတွင် ကန့်သတ်ထားသော *context window* ရှိသည်။ Coding agent များကို ထိရောက်စွာ အသုံးပြုနိုင်ရန် context ကို အကျိုးရှိစွာ အသုံးပြုဖို့ လိုအပ်သည်။ Agent ၌ လိုအပ်သော အချက်အလက်များ ရရှိနိုင်စေရေး သေချာအောင် ပြုလုပ်ရန် လိုအပ်သော်လည်း context window ပြည့်လျှံသွားခြင်း သို့မဟုတ် model ၏ စွမ်းဆောင်ရည် ကျဆင်းသွားခြင်းမှ ရှောင်ရှားရန် မလိုအပ်သော context များကို ရှောင်ကြဉ်ရမည် (context အရွယ်အစား ကြီးထွားလာသည်နှင့်အမျှ context window မပြည့်လျှံသေးသည့်တိုင် စွမ်းဆောင်ရည် ကျဆင်းသွားတတ်သည်)။ Agent harness များသည် context ကို အလိုအလျောက် ပံ့ပိုးပေးပြီး အတိုင်းအတာတစ်ခုအထိ စီမံခန့်ခွဲပေးသော်လည်း ထိန်းချုပ်မှု အများအပြားကိုမူ အသုံးပြုသူထံ ချန်ထားခဲ့သည်။
  - **Context window ကို ရှင်းလင်းခြင်း။** အခြေခံအကျဆုံး ထိန်းချုပ်မှုအဖြစ် coding agent များသည် context window ကို ရှင်းလင်းခြင်း (စကားပြောဆိုမှု အသစ်တစ်ခု စတင်ခြင်း) ကို ထောက်ပံ့ပေးပြီး၊ မသက်ဆိုင်သော မေးခွန်းများအတွက် ဤသို့ ပြုလုပ်သင့်သည်။
  - **စကားပြောဆိုမှုကို နောက်ပြန်ဆုတ်ခြင်း (Rewinding)။** အချို့သော coding agent များသည် စကားပြောဆိုမှု သမိုင်းကြောင်းရှိ အဆင့်များကို ပြန်ဖျက်ခြင်း (undoing) ကို ထောက်ပံ့ပေးသည်။ Agent အား အခြားလမ်းကြောင်းသို့ ဦးတည်စေမည့် ထပ်မံဖြည့်စွက် အကြောင်းကြားစာ ပေးပို့ခြင်းထက် “undo” ပြုလုပ်ခြင်းက ပိုမိုသင့်တော်သည့် အခြေအနေများတွင် ဤသို့ပြုလုပ်ခြင်းက context ကို ပိုမို ထိရောက်စွာ စီမံခန့်ခွဲနိုင်သည်။

Make up a quick demo.

- **Compaction ပြုလုပ်ခြင်း။** အကန့်အသတ်မရှိသော အရှည်ရှိ စကားပြောဆိုမှုများကို ဖြစ်နိုင်စေရန် coding agent များသည် context *compaction* ကို ထောက်ပံ့ပေးသည်- အကယ်၍ စကားပြောဆိုမှု သမိုင်းကြောင်းသည် လွန်စွာ ရှည်လျားလာပါက၊ စကားပြောဆိုမှု၏ ရှေ့ပိုင်းကို အကျဉ်းချုပ်ရန် LLM ကို အလိုအလျောက် ခေါ်ယူမည်ဖြစ်ပြီး၊ စကားပြောဆိုမှု သမိုင်းကြောင်းကို ထိုအကျဉ်းချုပ်ဖြင့် အစားထိုးမည်ဖြစ်သည်။ အချို့သော agent များသည် လိုအပ်သည့်အခါ compaction စေခိုင်းနိုင်ရန် အသုံးပြုသူထံ ထိန်းချုပ်ခွင့် ပေးထားသည်။

Show `/compact` in Claude Code, show full summary.

- **llms.txt။** `/llms.txt` ဖိုင်သည် inference ပြုလုပ်ချိန်တွင် LLM များ အသုံးပြုရန် ရည်ရွယ်သည့် document တစ်ခုအတွက် အဆိုပြုထားသော **standard** (https://llmstxt.org/) တည်နေရာ ဖြစ်သည်။ ထုတ်ကုန်များ (ဥပမာ **cursor.com/llms.txt** (https://cursor.com/llms.txt))၊ ဆော့ဖ်ဝဲလ် library များ (ဥပမာ **ai.pydantic.dev/llms.txt** (https://ai.pydantic.dev/llms.txt))၊ နှင့် API များ (ဥပမာ **apify.com/llms.txt** (https://apify.com/llms.txt)) တို့တွင် ဖွံ့ဖြိုးတိုးတက်ရေး လုပ်ငန်းများအတွက် အဆင်ပြေစေမည့် `llms.txt` ဖိုင်များ ရှိနိုင်သည်။ ထိုကဲ့သို့သော document များသည် token တစ်ခုစီအလိုက် သတင်းအချက်အလက် ပိုမို သိပ်သည်းသဖြင့် သင့် coding agent ထံ HTML စာမျက်နှာကို ရယူဖတ်ရှုခိုင်းခြင်းထက် context

အသုံးပြုမှု ပိုမို ထိရောက်သည်။ သင် အသုံးပြုရန် ကြိုးပမ်းနေသည့် dependency တစ်ခုနှင့် ပတ်သက်၍ coding agent တွင် တည်ဆောက်ပြီးသား ဗဟုသုတ မရှိသည့်အခါ (ဥပမာ LLM ၏ knowledge cutoff ပြီးမှ ထွက်ရှိလာခဲ့ခြင်း ကြောင့်) ပြင်ပ documentation များသည် အသုံးဝင်ပါသည်။

Side-by-side comparison in an empty repo (on Desktop or some other self-contained place, with `git init` run in it):

```
Write a single-file Python program example in demo.py using semlib to sort "Ily
a Sutskever", "Soumith Chintala", and "Donald Knuth" in terms of their fame as
AI researchers.

Write a single-file Python program example in demo.py using semlib to sort "Ily
a Sutskever", "Soumith Chintala", and "Donald Knuth" in terms of their fame as
AI researchers. See https://semlib.anish.io/llms.txt. Follow links to Markdown
versions of any pages linked in llms.txt files.
```

Not sure why the agent doesn't do this by default. You'd probably put that last sentence in a CLAUDE.md file.

- **AGENTS.md။** Coding agent အများစုသည် coding agent များအတွက် README အဖြစ် **AGENTS.md** (https://agents.md/) သို့မဟုတ် ဆင်တူရာ ဖိုင်များကို ထောက်ပံ့ပေးကြသည် (ဥပမာ Claude Code သည် `CLAUDE.md` ကို ရှာဖွေသည်)။ Agent စတင်သည့်အခါ ၎င်းသည် `AGENTS.md` ၏ ပါဝင်သည့် အချက်အလက် တစ်ခုလုံးဖြင့် context ကို ကြိုတင် ဖြည့်သွင်းပေးသည်။ Session များအားလုံး အတွက် အသုံးဝင်သည့် အကြံပြုချက်များကို agent သို့ ပေးအပ်ရန် ဤဖိုင်ကို အသုံးပြုနိုင်သည် (ဥပမာ code ပြောင်းလဲမှုများ ပြုလုပ်ပြီးနောက် type-checker ကို အမြဲ run ရန် သွန်ကြားခြင်း၊ unit test များ မည်သို့ run ရမည်ကို ရှင်းပြခြင်း သို့မဟုတ် agent ဝင်ရောက်ဖတ်ရှုနိုင်သည့် ပြင်ပ documentation လင့်ခ်များ ပံ့ပိုးပေးခြင်း)။ အချို့သော coding agent များသည် ဤဖိုင်ကို အလိုအလျောက် ထုတ်လုပ်ပေးနိုင်သည် (ဥပမာ Claude Code မှ `/init` command)။ `AGENTS.md` ၏ လက်တွေ့ကမ္ဘာ ဥပမာတစ်ခုကို ဤနေရာတွင် (https://github.com/pydantic/pydantic-ai/blob/main/CLAUDE.md) ကြည့်နိုင်ပါသည်။

Dotbot example, CLAUDE.md that includes @DEVELOPMENT.md and says to always run the type checker and code formatter after making any changes to Python code.

Example prompt, off of master:

```
Remove the "--version" command-line flag.
```

This is something that'll be fast, for demonstration purposes.

- **Skills များ။** `AGENTS.md` ထဲမှ အချက်အလက်များသည် agent ၏ context window ထဲသို့ အပြည့်အဝ အမြဲတမ်း load လုပ်ခံရသည်။ *Skills* များသည် context ပမာဏ မလိုအပ်ဘဲ ကြီးထွားလာခြင်း (context bloat) ကို ရှောင်ရှားရန် ကြားခံ အဆင့်တစ်ခုကို ပေါင်းစပ်ပေးသည်- သင်သည် agent အား skill များ၏ စာရင်းနှင့် ယင်းတို့၏ ဖော်ပြချက်များကို ပံ့ပိုးပေးနိုင်ပြီး၊ agent က လိုအပ်သည့်အခါမှသာ ထို skill ကို "ဖွင့်" နိုင်သည် (ယင်း၏ context window ထဲသို့ load လုပ်နိုင်သည်)။
- **Subagents များ။** အချို့သော coding agent များသည် သီးခြား task အလိုက် workflow များအတွက် agent များဖြစ်သည့် subagent များကို သတ်မှတ်ခွင့် ပြုထားသည်။ Top-level coding agent သည် သီးခြား task တစ်ခုကို ပြီးမြောက်စေရန် sub-agent တစ်ခုအား စေခိုင်းနိုင်ပြီး၊ ဤသည်မှာ top-level agent နှင့် subagent နှစ်ခုလုံးအတွက် context ကို ပိုမို ထိရောက်စွာ စီမံခန့်ခွဲနိုင်စေသည်။ Top-level agent ၏ context သည် subagent မြင်တွေ့သမျှ အရာအားလုံးဖြင့် ရှုပ်ထွေးမနေတော့ဘဲ၊ subagent သည်လည်း ယင်း၏ task အတွက် လိုအပ်သော context ကိုသာ ရရှိမည်ဖြစ်သည်။ ဥပမာတစ်ခုအနေဖြင့် အချို့သော coding agent များသည် web research ကို subagent အဖြစ် အကောင်အထည်ဖော်ကြသည်- top-level agent သည် subagent ထံ မေးခွန်းတစ်ခု မေးမည်ဖြစ်ပြီး၊ subagent က web ရှာဖွေခြင်းများ ပြုလုပ်၍ ဝက်ဘ်စာမျက်နှာတစ်ခုစီကို ရယူဖတ်ရှု သုံးသပ်ကာ မေးခွန်း၏ အဖြေကို top-level agent ထံ ပံ့ပိုးပေးမည်ဖြစ်သည်။ ဤနည်းဖြင့် top-level agent ၏ context တွင် ရယူထားသော ဝက်ဘ်စာမျက်နှာများ၏ အချက်အလက် အပြည့်အစုံကြောင့် ရှုပ်ထွေးမနေသလို၊ subagent ၏ context တွင်လည်း top-level agent ၏ ကျန်ရှိသော စကားပြောဆိုမှု သမိုင်းကြောင်းများ ပါဝင်မနေတော့ပါ။

Prompt များကို ရေးသားရန် လိုအပ်သည့် အဆင့်မြင့် feature အများအပြားအတွက် (ဥပမာ skills သို့မဟုတ် subagents)၊ စတင်နိုင်ရန် LLM များကို အသုံးပြုနိုင်သည်။ အချို့သော coding agent များတွင် ဤသို့ ပြုလုပ်နိုင်သည့် တည်ဆောက်ပြီးသား support ပင် ပါဝင်သည်။ ဥပမာ Claude Code သည် တိုတောင်းသော prompt တစ်ခုမှ subagent တစ်ခုကို ထုတ်လုပ်ပေးနိုင်သည် (`/agents` ကို ခေါ်ယူပြီး agent အသစ်တစ်ခု ဖန်တီးပါ)။ အောက်ပါ prompt ဖြင့် subagent တစ်ခု ဖန်တီးကြည့်ပါ-

```
A Python code checking agent that uses `mypy` and `ruff` to type-check, lint, a
nd format *check* any files that have been modified from the last git commit.
```

ထို့နောက် သင်သည် top-level agent အား "use the code checker subagent" ကဲ့သို့သော စာသားဖြင့် subagent အား တိုက်ရိုက် စေခိုင်းရန် အသုံးပြုနိုင်သည်။ သို့မဟုတ် သင့်တော်သည့်အခါတွင် top-level agent ကို subagent ထံ အလိုအလျောက် စေခိုင်းစေနိုင်သည်၊ ဥပမာ Python ဖိုင်များကို ပြင်ဆင်ပြီးနောက် မျိုး ဖြစ်သည်။

သတိပြုရမည့် အချက်များ (What to watch out for)

AI tool များသည် အမှားများ ပြုလုပ်နိုင်သည်။ ၎င်းတို့သည် နောက်လာမည့် token ကို ဖြစ်တန်းခြေအပေါ် မူတည်၍ ခန့်မှန်းပေးသည့် model အဖြစ်သာ တည်ဆောက်ထားသော LLM များပေါ်တွင် အခြေခံထားသည်။ ၎င်းတို့သည် လူသားများကဲ့သို့ "ဉာဏ်ရည်ထက်မြက်" သည် မဟုတ်ပါ။ မှန်ကန်မှုရှိမရှိနှင့် လုံခြုံရေးဆိုင်ရာ အမှားများ (security bugs) ရှိမရှိအတွက် AI ၏ output များကို စစ်ဆေးပါ။ အချို့အခါများတွင် code ကို စစ်ဆေးခြင်းသည် code ကို ကိုယ်တိုင် ရေးသားခြင်းထက် ပိုမို ခက်ခဲနိုင်သည်- အရေးကြီးသော code များအတွက် သင့်ကိုယ်တိုင် လက်ဖြင့် ရေးသားရန် စဉ်းစားပါ။ AI သည် လမ်းလွဲများသို့ ရောက်ရှိသွားနိုင်ပြီး သင့်အား မျက်စိလည်အောင် (gaslight) ကြိုးပမ်းနိုင်သည်- debugging သံသရာ (spirals) ထဲ ရောက်မသွားစေရန် သတိပြုပါ။ AI ကို အမှီသဟဲပြုစရာအဖြစ် မသုံးပါနှင့်၊ အလွန်အမင်း မှီခိုခြင်း သို့မဟုတ် အပေါ်ယံသာ နားလည်ခြင်းတို့ကို သတိထားပါ။ AI မပြုလုပ်နိုင်သေးသည့် ပရိုဂရမ်းမင်း task အမြောက်အမြား ရှိနေပါသေးသည်။ တွက်ချက်မှုဆိုင်ရာ စဉ်းစားတွေးခေါ်မှု (Computational thinking) သည် တန်ဖိုးရှိနေဆဲ ဖြစ်သည်။

အကြံပြုထားသော ဆော့ဖ်ဝဲလ်များ (Recommended software)

IDE များနှင့် AI coding extension အများအပြားတွင် coding agent များ ပါဝင်ကြသည် (**development environment သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/development-environment/) မှ အကြံပြုချက်များကို ကြည့်ပါ)။ အခြား လူကြိုက်များသော coding agent များတွင် Anthropic ၏ **Claude Code** (https://www.claude.com/product/claude-code)၊ OpenAI ၏ **Codex** (https://openai.com/codex/) နှင့် **opencode** (https://github.com/anomalyco/opencode) ကဲ့သို့သော open-source agent များ ပါဝင်သည်။

လေ့ကျင့်ခန်းများ (Exercises)

1. ပရိုဂရမ်းမင်း task တစ်ခုတည်းကို လေးကြိမ် ပြုလုပ်ခြင်းဖြင့် ကိုယ်တိုင် လက်ဖြင့် ကုဒ်ရေးခြင်း၊ AI autocomplete အသုံးပြုခြင်း၊ inline chat အသုံးပြုခြင်းနှင့် agent များ အသုံးပြုခြင်းတို့၏ အတွေ့အကြုံများကို နှိုင်းယှဉ်ကြည့်ပါ။ အကောင်းဆုံး ရွေးချယ်မှုမှာ သင် လုပ်ဆောင်နေဆဲဖြစ်သော project မှ သေးငယ်သော feature တစ်ခု ဖြစ်သည်။ အကယ်၍ အခြား အိုင်ဒီယာများကို ရှာဖွေနေပါက GitHub ရှိ open-source project များမှ “good first issue” ပုံစံ task များကို ပြီးမြောက်အောင် လုပ်ဆောင်ခြင်း သို့မဟုတ် **Advent of Code** (https://adventofcode.com/) သို့မဟုတ် **LeetCode** (https://leetcode.com/) ပုစ္ဆာများကို ဖြေရှင်းခြင်းဖြင့် စဉ်းစားနိုင်သည်။
2. မရင်းနှီးသေးသော codebase တစ်ခုကို လေ့လာစူးစမ်းရန် AI coding agent တစ်ခုကို အသုံးပြုပါ။ ဤသည်ကို သင် အမှန်တကယ် စိတ်ဝင်စားသော project တစ်ခုတွင် debug ပြုလုပ်လိုခြင်း သို့မဟုတ် feature အသစ်တစ်ခု ထည့်သွင်းလိုခြင်း အခြေအနေတွင် အကောင်းဆုံး ပြုလုပ်နိုင်သည်။ အကယ်၍ စိတ်ထဲတွင် မရှိပါက **opencode** (https://github.com/anomalyco/opencode) agent တွင် လုံခြုံရေးဆိုင်ရာ feature များ မည်သို့ အလုပ်လုပ်သည်ကို နားလည်ရန် AI agent တစ်ခုကို အသုံးပြုကြည့်ပါ။
3. အစမှစ၍ သေးငယ်သော app တစ်ခုကို vibe code ပြုလုပ်ကြည့်ပါ။ သင့်ကိုယ်တိုင် လက်ဖြင့် code စာကြောင်းတစ်ကြောင်းမျှ မရေးပါနှင့်။
4. သင် ရွေးချယ်ထားသော coding agent အတွက် `AGENTS.md` (သို့မဟုတ် သင့် agent ၏ ဆင်တူရာ ဥပမာ `CLAUDE.md`)၊ skill တစ်ခု (ဥပမာ **skill in Claude Code** (https://code.claude.com/docs/en/skills) သို့မဟုတ် **skill in Codex** (https://developers.openai.com/codex/skills/))၊ နှင့် subagent တစ်ခု (ဥပမာ **subagent in Claude Code** (https://code.claude.com/docs/en/sub-agents)) တို့ကို ဖန်တီး၍ စမ်းသပ်ပါ။ ၎င်းတို့ထဲမှ တစ်ခုကို အခြားတစ်ခုအစား မည်သည့်အချိန်တွင် အသုံးပြုချင်မည်နည်းဆိုသည်ကို စဉ်းစားပါ။ သင် ရွေးချယ်ထားသော coding agent သည် ဤ လုပ်ဆောင်ချက်အချို့ကို ထောက်ပံ့မည် မဟုတ်သည်ကို သတိပြုပါ- ၎င်းတို့ကို ကျော်လွန်သွားနိုင်သည် သို့မဟုတ် ထောက်ပံ့မှုပေးသော အခြား coding agent တစ်ခုကို စမ်းသပ်ကြည့်နိုင်သည်။
5. **Code Quality သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/code-quality/) မှ Markdown bullet point များ regex လေ့ကျင့်ခန်းအတိုင်း တူညီသော ရည်မှန်းချက် ပြည့်မြောက်စေရန် coding agent တစ်ခုကို အသုံးပြုပါ။ ၎င်းသည် ဖိုင်ကို တိုက်ရိုက် ပြင်ဆင်ခြင်းဖြင့် task များကို ပြီးမြောက်စေသလား။ ဤကဲ့သို့သော task မျိုးကို ပြီးမြောက်ရန် ဖိုင်ကို Agent က တိုက်ရိုက် ပြင်ဆင်ခြင်း၏ အားနည်းချက်များနှင့် ကန့်သတ်ချက်များမှာ မည်သည်တို့နည်း။ ဖိုင်အား တိုက်ရိုက် ပြင်ဆင်ခြင်း မဟုတ်ဘဲ ပြီးမြောက်အောင် လုပ်ဆောင်နိုင်ရန် Agent ထံ မည်သို့ prompt ပေးရမည်နည်းဆိုသည်ကို ရှာဖွေပါ။ အကူအညီ- **ပထမ**

[ဦးဆုံး သင်ခန်းစာ](https://missing-semester-my.github.io/2026/course-shell/) တွင် ဖော်ပြခဲ့သော command-line tool များထဲမှ တစ်ခုကို အသုံးပြုရန် agent ထံ တောင်းဆိုပါ။

6. Coding agent အများစုသည် “yolo mode” ပုံစံကို ထောက်ပံ့ပေးကြသည် (ဥပမာ Claude Code တွင် `--dangerously-skip-permissions`)။ ဤ mode ကို တိုက်ရိုက် အသုံးပြုခြင်းသည် လုံခြုံမှုမရှိသော်လည်း virtual machine သို့မဟုတ် container ကဲ့သို့ သီးခြားခွဲထုတ်ထားသော ဝန်းကျင် (isolated environment) တွင် coding agent တစ်ခုကို run ပြီး ကိုယ်ပိုင် လုပ်ဆောင်မှု (autonomous operation) ကို ဖွင့်ပေးခြင်းသည် လက်ခံနိုင်ဖွယ် ရှိသည်။ သင့် စက်ပေါ်တွင် ဤ setup ကို run နိုင်အောင် ပြုလုပ်ပါ။ **Claude Code devcontainers** (https://code.claude.com/docs/en/devcontainer) သို့မဟုတ် **Docker Sandboxes / Claude Code** (https://docs.docker.com/ai/sandboxes/agents/claude-code/) ကဲ့သို့သော documentation များသည် အသုံးဝင်နိုင်ပါသည်။ ဤ setup ကို တည်ဆောက်ရန် နည်းလမ်းတစ်မျိုးထက်မက ရှိပါသည်။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. Development Environment and Tools	https://missing-semester-my.github.io/2026/development-environment/	
2. AI-powered development	https://missing-semester-my.github.io/2026/development-environment/#ai-powered-development	
3. large language models (LLMs)	https://en.wikipedia.org/wiki/Large_language_model	
4. code စာကြောင်းရေ ၂၀၀ ဖြင့် ရေးသားအကောင်အထည်ဖော်နိုင်သည်	https://www.mihaileric.com/The-Emperor-Has-No-Clothes/	
5. အကောင်အထည်ဖော်ရန်	https://github.com/missing-semester/missing-semester/pull/345	
6. f552b55	https://github.com/missing-semester/missing-semester/commit/f552b5523462b22b8893a8404d2110c4e59613dd	
7. f1e1c41	https://github.com/missing-semester/missing-semester/commit/f1e1c417adba6b4149f7eef91ff5624de40dc637	
8. code intelligence	https://missing-semester-my.github.io/2026/development-environment/#code-intelligence-and-language-servers	
9. ခွဲထုတ်ရန်	https://github.com/missing-semester/missing-semester/pull/344	
10. GitHub CLI	https://cli.github.com/	
11. ဥပမာတစ်ခုကို ဤနေရာတွင် ကြည့်နိုင်ပါသည်	https://github.com/cleanlab/office-presence-dashboard	
12. skills	https://code.claude.com/docs/en/skills	
13. အစားထိုးလိုက်ပါပြီ	https://developers.openai.com/codex/custom-prompts	
14. version control	https://missing-semester-my.github.io/2026/version-control/	
15. git worktrees	https://git-scm.com/docs/git-worktree	

Resource / Description	URL	Micro QR
16. Notion MCP server	https://github.com/makenotion/notion-mcp-server	
17. Pulse	https://www.pulsemcp.com/servers	
18. Glama	https://glama.ai/mcp/servers	
19. standard	https://llmstxt.org/	
20. cursor.com/llms.txt	https://cursor.com/llms.txt	
21. ai.pydantic.dev/llms.txt	https://ai.pydantic.dev/llms.txt	
22. apify.com/llms.txt	https://apify.com/llms.txt	
23. AGENTS.md	https://agents.md/	
24. ဤနေရာတွင်	https://github.com/pydantic/pydantic-ai/blob/main/CLAUDE.md	
25. Claude Code	https://www.claude.com/product/claude-code	
26. Codex	https://openai.com/codex/	
27. opencode	https://github.com/anomalyco/opencode	
28. Advent of Code	https://adventofcode.com/	
29. LeetCode	https://leetcode.com/	
30. skill in Codex	https://developers.openai.com/codex/skills/	
31. subagent in Claude Code	https://code.claude.com/docs/en/sub-agents	

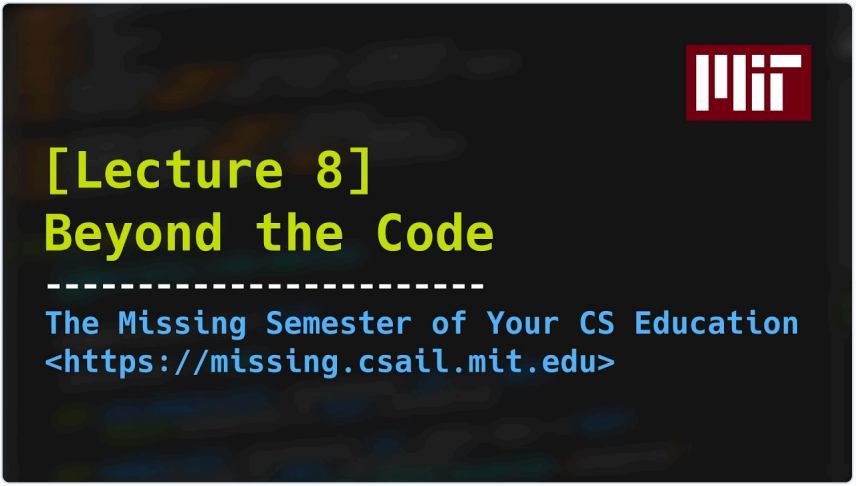
Resource / Description	URL	Micro QR
32. Code Quality သင်ခန်းစာ	https://missing-semester-my.github.io/2026/code-quality/	
33. ပထမဦးဆုံး သင်ခန်းစာ	https://missing-semester-my.github.io/2026/course-shell/	
34. Claude Code devcontainers	https://code.claude.com/docs/en/devcontainer	
35. Docker Sandboxes / Claude Code	https://docs.docker.com/ai/sandboxes/agents/claude-code/	

Beyond the code (ကုဒ်ရေးသားခြင်းထက် ကျော်လွန်၍)

အနှစ်ချုပ် - Documentation ရေးသားခြင်း၊ အိုးပင်းဆော့စ် အသိုင်းအဝိုင်း ကျင့်ဝတ်များ၊ နှင့် AI အသုံးပြုမှု ကျင့်ဝတ်များ အပါအဝင် မရှိမဖြစ် လိုအပ်သော ဆော့ဖ်စကေးလ်များ (soft skills) အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Beyond the code (ကုဒ်ရေးသားခြင်းထက် ကျော်လွန်၍)



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=2DOEATfXT8k

ကောင်းမွန်သော ဆော့ဖ်ဝဲ အင်ဂျင်နီယာတစ်ဦး ဖြစ်လာရန်မှာ အလုပ်လုပ်သည့် ကုဒ် ရေးသားနိုင်ခြင်း တစ်ခုတည်း မကပါဘူး။ မိမိကိုယ်တိုင် အပါအဝင် အခြားသူများ (နောင်တစ်ချိန်တွင် ကြည့်မည့် မိမိ အပါအဝင်) နားလည်နိုင်၊ ထိန်းသိမ်းပြင်ဆင်နိုင်၊ ၎င်းအပေါ်တွင် ထပ်မံ တည်ဆောက်နိုင်မည့် ကုဒ်မျိုးကို ရေးသားနိုင်ခြင်းလည်း ဖြစ်ပါတယ်။ ထို့အပြင် ရှင်းလင်းစွာ ဆက်သွယ်ပြောဆိုခြင်း၊ သေချာ စဉ်းစားသုံးသပ်၍ ပါဝင်ကူညီခြင်း၊ နှင့် အိုးပင်းဆော့စ်ဖြစ်စေ၊ ပရိုပရိုက်ထရီ (proprietary) ဖြစ်စေ မိမိ ပါဝင်လှုပ်ရှားနေသော အသိုင်းအဝိုင်းဂေဟစနစ်တွင် နိုင်ငံသားကောင်းတစ်ဦး သဖွယ် ပြုမူနေထိုင်ခြင်းတို့လည်း ပါဝင်ပါတယ်။

တစ်လမ်းသွား ဆက်သွယ်ပြောဆိုမှု (One-way communication)

ဆော့ဖ်ဝဲ အင်ဂျင်နီယာ လုပ်ငန်းစဉ်များစွာတွင် လက်ရှိ သင်၏ အကြောင်းအရင်းအချက်အလက် (context) များကို မသိရှိကြသူများအတွက် ရေးသားရခြင်းများ ပါဝင်လေ့ရှိပါတယ် - နောက်မှ အဖွဲ့ထဲ ရောက်လာသည့် အဖွဲ့ဝင်များ၊ သင်၏ ကုဒ်ကို ဆက်လက် ထိန်းသိမ်းရမည့် ထိန်းသိမ်းသူများ (maintainers)၊ သို့မဟုတ် အချို့သော ရွေးချယ်မှုများကို အဘယ်ကြောင့် ပြုလုပ်ခဲ့သလဲဆိုသည်ကို မေ့သွားသည့် လွန်ခဲ့သော ၆ လက မိမိကိုယ်တိုင်တို့ ဖြစ်ကြပါတယ်။ ဤသို့သော စာရေးသားခြင်း အမျိုးအစား အားလုံးအတွက် အဓိက အကြံပြုချက်မှာ သင်၏ ရည်မှန်းချက်သည် *ဘာလုပ်ထားသလဲ* (*what*) ဆိုသည်တင်မကဘဲ *အဘယ်ကြောင့် ပြုလုပ်ရသလဲ* (*why*) ဆိုသည်ကိုပါ မှတ်တမ်းတင် ချပြရန် ဖြစ်ပါတယ်။ *ဘာလုပ်ထားသလဲ* ဆိုသည်မှာ ကုဒ်ကို ကြည့်ရုံဖြင့် သဘောပေါက်နိုင်သော်လည်း *အဘယ်ကြောင့် ပြုလုပ်ရသလဲ* ဆိုသည်မှာမူ အချိန်ကြာလာသည်နှင့်အမျှ လွယ်ကူစွာ မေ့ပျောက်သွားနိုင်သော ခက်ခဲစွာ ရရှိထားသည့် အသိပညာဖြစ်ပါတယ်။

အင်ဂျင်နီယာအချင်းချင်း ဆက်သွယ်ပြောဆိုသည့် အလေ့အထများအနက် (ကုဒ် သီးသန့် မဟုတ်ပါက) အသုံးအများဆုံး နည်းလမ်းမှာ ကုဒ် ကွန်မန့်များ (code comments) ဖြစ်နိုင်ပါတယ်။ ကျွန်ုပ်၏ ကိုယ်တွေ့အရ အချို့သော ကုဒ်ကွန်မန့်များသည် အသုံးမဝင်သည်ကို တွေ့ရပါတယ်။ သို့သော် ၎င်းတို့သည် အမြဲတမ်း အသုံးမဝင်ဘဲ မနေသင့်ပါဘူး။ ကောင်းမွန်သော ကွန်မန့်များသည် ကုဒ်ကိုယ်တိုင်က မရှင်းပြနိုင်သော အရာများကို ရှင်းပြပေးကြပါတယ် - ဆိုလိုသည်မှာ အရာတစ်ခုကို အဘယ်ကြောင့် ဤနည်းလမ်းအတိုင်း ပြုလုပ်ခဲ့သနည်း (*why*) ဆိုသည်ကို ရှင်းပြခြင်းဖြစ်ပြီး၊ ၎င်းက မည်သို့ အလုပ်လုပ်သနည်း (*how*) ဆိုသည်ကို မဟုတ်ပါ (၎င်းကို ကုဒ်က ပြသထားပြီး ဖြစ်ပါတယ်)။ ၎င်းတို့သည် နာရီပေါင်းများစွာ ကြာမြင့်နိုင်သော ဇဝေဇဝါ ဖြစ်မှုများကို သက်သာစေနိုင်ပြီး၊ မကောင်းသော ကွန်မန့်များသည် အနှောင့်အယှက် ဖြစ်စေနိုင်သလို၊ ပိုဆိုးသည်မှာ လမ်းလွဲ ရောက်စေနိုင်ပါတယ်။

နီးပါးမျှ အမြဲတမ်း တန်ဖိုးရှိသည့် ကွန်မန့် အမျိုးအစားများမှာ -

- **TODO များ**: မပြီးသေးသော သို့မဟုတ် မသပ်ရပ်သေးသော ကုဒ်များကို မှတ်သားထားပါ၊ သို့သော် အခြားသူတစ်ဦးအနေဖြင့် မည်သည့်အရာ ကျန်ရှိနေသနည်းနှင့် အဘယ်ကြောင့် ဆိုင်းငံ့ထားခဲ့သနည်း ဆိုသည်ကို နားလည်နိုင်ရန် လုံလောက်သော အကြောင်းအရင်း (context) ကို ချန်ထားခဲ့ပါ။ “TODO: optimize” ဟု ရေးခြင်းသည် အသုံးမဝင်ပါ၊ “TODO: ဤ O(n²) loop သည် `n<100` အတွက် အဆင်ပြေသော်လည်း Scale လုပ်ပါက indexing လိုအပ်ပါလိမ့်မည်” ဟု ရေးခြင်းက အမှန်တကယ် ဆောင်ရွက်နိုင်သော လမ်းညွှန်ချက် ဖြစ်ပါတယ်။
- **ကိုးကားချက်များ (References)**: ကုဒ်သည် သုတေသနစာတမ်းပါ အယ်လဂိုရီသမ်ကို အကောင်အထည်ဖော်ခြင်း၊ အခြားနေရာမှ ကုဒ်ကို မှီငြမ်းပြင်ဆင်ခြင်း သို့မဟုတ် Documentation ပါ

သတ်မှတ်ချက်အတိုင်း မူတည်ရေးသားခြင်းတို့ ပြုလုပ်ပါက ပြင်ပ အရင်းအမြစ်များသို့ လင့်ခ်ချိတ်ဆက်ပေးပါ။ Permalinks များကို အသုံးပြုပါ။ မူရင်းကိုးကားချက်မှ ခွဲထွက်ပြောင်းလဲသွားသည့် အချက်များရှိပါက မှတ်သားဖော်ပြပါ။

- **မှန်ကန်ကြောင်း အထောက်အထားများ (Correctness arguments)**: ရှုပ်ထွေးသော ကုဒ်များသည် အဘယ်ကြောင့် မှန်ကန်သော ရလဒ်များ ထွက်ပေါ်စေသည်ဆိုသည့် *အကြောင်းအရင်း*ကို ရှင်းပြပါ။ ကုဒ်သည် အဆင့်များကို ဖော်ပြပေးပြီး ကွန်မန့်က ထိုအဆင့်များ အဘယ်ကြောင့် အလုပ်လုပ်သည်ကို ရှင်းပြပေးပါတယ်။
- **ခက်ခဲစွာ သင်ယူခဲ့ရသော သင်ခန်းစာများ (Hard-learned lessons)**: အကယ်၍ သင်သည် အရာတစ်ခုကို ဖြေရှင်းရန် (debugging) မိနစ် ၃၀ ကျော် သုံးခဲ့ရပြီး ယင်းဖြေရှင်းချက်မှာ မသိသာသော နည်းလမ်းဖြစ်ပါက မှတ်တမ်းတင်ထားပါ။ လွန်ခဲ့သော သင်ကိုယ်တိုင်က ၎င်းလိုအပ်သည်ကို သတိမပြုမိခဲ့သလို၊ နောင်တွင် ဖတ်ရှုမည့်သူများလည်း သတိပြုမိမည် မဟုတ်ပါ။
- **ကိန်းသေများအတွက် အကြောင်းပြချက်များ (Rationale for constants)**: သီးသန့် ကိန်းဂဏန်းများ (Magic numbers) သည် ရှင်းလင်းချက် လိုအပ်ပါတယ်။ အဘယ်ကြောင့် 1492 နံပါတ် ဖြစ်ရသနည်း။ အဘယ်ကြောင့် 16 bits ဖြစ်ရသနည်း။ ၎င်းကို ကြံုရာ ရွေးချယ်ခဲ့ခြင်းလား၊ စမ်းသပ်မှုမှ ထွက်ပေါ်လာခြင်းလား၊ သို့မဟုတ် မှန်ကန်မှုအတွက် လိုအပ်၍လား။ “ကြံုရာ ရွေးချယ်ထားခြင်း” ဆိုသည့် စကားပင် လျှင် အသုံးဝင်သော သတင်းအချက်အလက် ဖြစ်ပါတယ်။
- **အရေးပါသော ရွေးချယ်မှုများ (Load-bearing choices)**: အကယ်၍ စနစ်မှန်ကန်မှုသည် မသိသာသော ရေးသားမှု အသေးစိတ်တစ်ခုပေါ်တွင် မူတည်နေပါက (ဥပမာ - “အောက်ပါအတိုင်း ပတ်လည်စစ်ဆေးမှု (iteration order) အစီအစဉ် အရေးကြီးသောကြောင့် BTreeSet ဖြစ်ရပါမည်”) ၎င်းကို အတိအလင်း ဖော်ပြထားပါ။
- **“အဘယ်ကြောင့် မသုံးသနည်း” များ (“Why not”s)**: ထင်ရှားသော နည်းလမ်းကို သင်တမင် ရှောင်ရှားခဲ့ပါက အဘယ်ကြောင့် ရှောင်ရှားခဲ့သနည်းဆိုသည်ကို ရှင်းပြပါ။ သို့မဟုတ်ပါက အခြားသူတစ်ဦးက နောင်တွင် လာရောက် “ပြင်ဆင်” ပြီး အရာအားလုံးကို ပျက်စီးစေပါလိမ့်မည်။

README များသည်လည်း (သင့်ထံတွင် တစ်ခု ရှိတယ်မလား) အခြားသော Developer များနှင့် ပထမဆုံး ထိတွေ့ဆက်ဆံသည့် အသုံးများသော နေရာတစ်ခု ဖြစ်ပါတယ်။ ကောင်းမွန်သော README တစ်ခုသည် မေးခွန်း လေးခုကို ချက်ချင်း အဖြေပေးနိုင်ပါတယ် - ၎င်းသည် ဘာလုပ်သနည်း၊ အဘယ်ကြောင့် စိတ်ဝင်စားသင့်သနည်း၊ မည်သို့ အသုံးပြုရသနည်း၊ မည်သို့ တပ်ဆင်ရသနည်း။ ဤအစီအစဉ်အတိုင်း အဖြေပေးရပါမည်။ ၎င်းကို ကတော့ (funnel) ပုံစံမျိုး တည်ဆောက်ပါ - ထိပ်ဆုံးတွင် စာတစ်ကြောင်း ရှင်းလင်းချက်နှင့် ရုပ်မြင်သရုပ်ပြ Demo ပါဝင်ကောင်း ပါဝင်နိုင်ပြီး လူတစ်ယောက်အနေဖြင့် ၎င်းသည် သူတို့၏ ပြဿနာကို ဖြေရှင်းပေးနိုင်သလားဆိုသည်ကို စက္ကန့်ပိုင်းအတွင်း ဆုံးဖြတ်နိုင်အောင် ပြုလုပ်ပါ။ ထို့နောက်မှ အသေးစိတ်များကို အဆင့်ဆင့် ထည့်သွင်းသွားပါ။ တပ်ဆင်နည်း (installation) မပြမီ အသုံးပြုပုံ (usage) ကို ပြသပါ - လူများသည် တပ်ဆင်မှု အဆင့်များကို မပြုလုပ်မီ သူတို့ မည်သည့်အရာ ရရှိမည်ကို ကြည့်ရှုချင်ကြပါတယ်။

Commit မက်ဆေ့ဂျ်များသည် မကြာခဏ လျစ်လျူရှုခံရလေ့ရှိသော အခြားသော “အခြားသူများအတွက် ရေးသားခြင်း” အမျိုးအစားတစ်ခု ဖြစ်ပါတယ်။ ၎င်းတို့ကို “fixed blah” သို့မဟုတ် “added foo” ဟူ၍ ရေးသားလေ့ရှိကြပြီး၊ အချို့ကိစ္စများတွင် လုံလောက်နိုင်သော်လည်း၊ ၎င်းတို့သည် အဘယ်ကြောင့် ကုဒ် အစုအဝေး (codebase) ဤသို့ ပြောင်းလဲလာရသနည်းဆိုသည့် *အကြောင်းအရင်း* မှတ်တမ်းအဖြစ် တည်ရှိနေသည်ကို မေ့လျော့သွားတတ်ကြပါတယ်။ တစ်စုံတစ်ယောက် (သင်ကိုယ်တိုင် အပါအဝင်) က ရှုပ်ထွေးသော ပြောင်းလဲမှုကို နားလည်ရန် `git blame` ကို စစ်ဆေးသည့်အခါ ကောင်းမွန်သော commit မက်ဆေ့ဂျ်များသည် အဖြေပေးနိုင်ရပါမည်။

ယေဘုယျအားဖြင့်၊ မက်ဆေ့ဂျ်၏ အဓိက စာကိုယ်သည် အောက်ပါမေးခွန်းများကို အဖြေပေးသင့်ပါတယ် - - မည်သည့် ပြဿနာက ဤပြောင်းလဲမှုကို ပြုလုပ်ရန် ဖိအားပေးခဲ့သနည်း။ - မည်သည့် အခြားရွေးချယ်စရာနည်းလမ်းများကို သင် စဉ်းစားခဲ့သနည်း။ - ရလဒ် သို့မဟုတ် ရင်းနှီးပေးဆပ်ရမှု (trade-offs) များသည် မည်သည်တို့ ဖြစ်သနည်း။ - ဤနည်းလမ်းနှင့် ပတ်သက်၍ အံ့အားသင့်ဖွယ် ဖြစ်နိုင်သည်မှာ မည်သည်နည်း။

> သိသာထင်ရှားသည်မှာ အသေးစိတ် ရေးသားချက်ကို ရှုပ်ထွေးမှုနှင့်အညီ ချိန်ဆရပါမည်။ စာလုံးပေါင်း အမှားပြင်ခြင်းကဲ့သို့ ရိုးရှင်းသော ပြောင်းလဲမှုအတွက် ခေါင်းစဉ် (subject) တစ်ကြောင်းသာ လိုအပ်ပါလိမ့်မည်။ Debug ပြုလုပ်ရန် နာရီပေါင်းများစွာ သုံးခဲ့ရသော သိမ်မွေ့သည့် race condition ပြုပြင်မှုမျိုးတွင်မူ ပြဿနာနှင့် ဖြေရှင်းချက်ကို ရှင်းပြထားသည့် စာပိုဒ်များ ရေးသားရန် ထိုက်တန်ပါတယ်။

ရှုပ်ထွေးသော ပြောင်းလဲမှုများအတွက် ပြဿနာ (Problem) → ဖြေရှင်းချက် (Solution) → နောက်ဆက်တွဲ ရလဒ်များ (Implications) အဆင်လိုက် တည်ဆောက်ပုံကို လိုက်နာခြင်းက အသုံးဝင်နိုင်ပါတယ် - ဖိအားပေးသည့် အကြောင်းအရင်း သို့မဟုတ် ကန့်သတ်ချက်ဖြင့် စတင်ပါ၊ ထို့နောက် မည်သည်တို့ ပြောင်းလဲသွားသည်နှင့် အဓိက ဒီဇိုင်း ဆုံးဖြတ်ချက်များကို ရှင်းပြပါ၊ ထို့နောက် သတိပြုဖွယ် ရလဒ်များ (ကောင်းကျိုးနှင့် ဆိုးကျိုးများ) ကို စာရင်းပြုလုပ်ပါ။ နောက်ဆုံးအပိုင်းသည် အထူးပင် အရေးကြီးပါတယ် - စစ်မှန်သော အင်ဂျင်နီယာ လုပ်ငန်းသည် ကောင်းကျိုးဆိုးကျိုးများကို မျှတအောင် ချိန်ဆရခြင်း ပါဝင်ပြီး ရင်းနှီးပေးဆပ်ရမှု (trade-off) ကို တမင်တကာ ပြုလုပ်ခဲ့ခြင်းဖြစ်ကြောင်း မှတ်တမ်းတင်ခြင်းက နောင်လာမည့် Developer များကို သင် ပြဿနာကို မမြင်ဘဲ လျစ်လျူရှုခဲ့သည်ဟု ထင်မြင်ခြင်းမှ ကာကွယ်ပေးပါတယ်။

LLM များကို Commit မက်ဆေ့ဂျ်များ ရေးသားရာတွင် အကူအညီ ရယူ *နိုင်ပါသည်*။ သို့သော် အကယ်၍ သင်သည် ပြောင်းလဲမှုကို LLM သို့ ပြသပြီး Commit မက်ဆေ့ဂျ် ရေးခိုင်းရုံမျှဖြင့် LLM သည် *ဘာလုပ်ထားသလဲ* (*what*) ဆိုသည်ကိုသာ ရရှိနိုင်မည်ဖြစ်ပြီး၊ *အဘယ်ကြောင့် ပြုလုပ်သနည်း* (*why*) ဆိုသည်ကို ရရှိမည် မဟုတ်ပါ။ ထို့ကြောင့် ထွက်ပေါ်လာသော Commit မက်ဆေ့ဂျ်သည် ဖော်ပြချက် သက်သက်သာ ဖြစ်နေပါလိမ့်မည် (ကျွန်ုပ်တို့ လိုချင်သည့် အရာနှင့် ဆန့်ကျင်ဘက်ဖြစ်သည်!)။ အကယ်၍ သင်သည် ပြောင်းလဲမှုကို ပြုလုပ်ရန် LLM ကို အသုံးပြုခဲ့ပါက၊ ထို LLM နှင့် ဆွေးနွေးနေသည့် Session တွင်းမှာပင် Commit မက်ဆေ့ဂျ် ရေးခိုင်း

ခြင်းက ပိုမို ကောင်းမွန်သော နည်းလမ်းဖြစ်နိုင်ပါတယ်၊ အကြောင်းမှာ LLM နှင့် ဆွေးနွေးထားချက်များသည် ပြောင်းလဲမှုဆိုင်ရာ အကြောင်းအရင်းများ (context) ကြွယ်ဝစွာ ပါဝင်နေသောကြောင့် ဖြစ်ပါတယ်။ သို့မဟုတ် ပါက (သို့မဟုတ် ထပ်ဆောင်းအနေဖြင့်) အသုံးဝင်သော နည်းလမ်းတစ်ခုမှာ LLM အား "အဘယ်ကြောင့် ပြုလုပ်သနည်း" (*why*) ကို အဓိကထားသည့် (နှင့် အထက်ပါ မှတ်စုများပါ သိမ်မွေ့သော အချက်များ ပါဝင်သည့်) Commit မက်ဆေ့ဂျ်မျိုး လိုချင်ကြောင်း အတိအလင်း ပြောကြားပြီး *လိုအကွေ့ငျးအရာမြားအတှကျ မေးမြန်းခိုငျးခွငျး* ဖြစ်ပါတယ်။ အခြေခံအားဖြင့် သင်သည် Coding Agent အတွက် အကြောင်းအရင်းများကို "ဖတ်ရှုနိုင်သော" MCP "tool" တစ်ခုကဲ့သို့ ဆောင်ရွက်ပေးနေခြင်း ဖြစ်ပါတယ်။

သင်၏ ပြောင်းလဲမှုများ ပိုမို ရှုပ်ထွေးလာသည်နှင့်အမျှ Commit များကို သုတ္တတြိယ (logically) ခွဲခြားရန် မမေ့ပါနှင့် (`git add -p` သည် သင်၏ မိတ်ဆွေဖြစ်ပါတယ်။)။ Commit တိုင်းသည် သီးခြားစီ နားလည်နိုင်ပြီး ပြန်လည်သုံးသပ်နိုင်သော ခိုင်မာသည့် ပြောင်းလဲမှုတစ်ခုကို ကိုယ်စားပြုရပါမည်။ Refactoring ပြုလုပ်ခြင်းကို အင်္ဂါရပ်အသစ်များ (new features) နှင့် ရောနှောခြင်း သို့မဟုတ် ဆက်စပ်မှုမရှိသော bug ပြုပြင်မှုများကို ပေါင်းစပ်ခြင်း မပြုပါနှင့်၊ ၎င်းသည် မည်သည့် ပြောင်းလဲမှုက မည်သည့် ပြဿနာကို ဖြေရှင်းခဲ့သလဲ ဆိုသည်ကို ရှုပ်ထွေးစေပြီး သင်၏ ပြောင်းလဲမှုများကို ပြန်လည် သုံးသပ်သည့်အခါ နှောင့်နှေးစေမည်မှာ သေချာသလောက် ရှိပါတယ်။ ထို့အပြင် ၎င်းသည် `git bisect` မှတစ်ဆင့် အလွန်အသုံးဝင်သော စွမ်းရည်များကို ပေးစွမ်းနိုင်သော်လည်း၊ ယင်းမှာ အခြားအချိန်မှ ပြောရမည့် အကြောင်းအရာ ဖြစ်ပါတယ်။

> နည်းပညာဆိုင်ရာ စာရေးသားခြင်းများတွင် ပိုမို ဂရုစိုက်လာပြီး ကျယ်ကျယ်ပြန့်ပြန့် အသုံးပြုလာသည်နှင့်အမျှ စာဖတ်သူကို လေးစားမှုရှိရန် မမေ့ပါနှင့်။ စတင်လိုက်သည်နှင့် အသေးစိတ် လွန်ကဲစွာ ရှင်းပြလိုသည့် စိတ်ဖြစ်ပေါ်လာရန် လွယ်ကူသော်လည်း စာဖတ်သူက သင်ရေးထားသည်ကို *လုံး၀* မဖတ်ဘဲ မနေစေရန် ထိုဆန္ဒကို ထိန်းချုပ်ရပါမည်။ *အဘယ်ကြောင့်* (*why*) ကို ရှင်းပြပါ၊ ထို့နောက် သူတို့၏ အခြေအနေအတွက် *မည်သို့ပြုလုပ်ရမည်နည်း* (*how*) ကို သူတို့ကိုယ်တိုင် ရှာဖွေနိုင်မည်ဟု ယုံကြည်ပါ။

ပူးပေါင်းဆောင်ရွက်ခြင်း (Collaboration)

အင်ဂျင်နီယာများအနေဖြင့် ကျွန်ုပ်တို့သည် လုပ်ငန်းခွင်၏ အချိန်အများစုကို မိမိတို့ ကီးဘုတ်တွင် ကုဒ် ရေးသားခြင်းဖြင့် ကုန်လွန်စေသော်လည်း အချိန် တော်တော်များများကိုမူ အခြားသူများနှင့် ဆက်သွယ် ပြောဆိုခြင်းဖြင့် ကုန်လွန်စေရပါတယ်။ ထိုအချိန်ကို ပူးပေါင်းဆောင်ရွက်ခြင်း (collaboration) နှင့် လေ့လာ သင်ယူခြင်း/မျှဝေခြင်း (education) ဟူ၍ ခွဲခြားလေ့ရှိပြီး နှစ်ခုစလုံးတွင် ပိုမို တိုးတက်အောင် ရင်းနှီးမြှုပ်နှံ ခြင်း၏ အကျိုးကျေးဇူးမှာ အလွန်ပင် ကြီးမားပါတယ်။

ပါဝင်ကူညီဆောင်ရွက်ခြင်း (Contributing)

သင်သည် Bug အစီရင်ခံစာ ပေးပို့ခြင်း၊ ရိုးရှင်းသော Bug ပြုပြင်မှု ပါဝင်ကူညီခြင်း သို့မဟုတ် ကြီးမားသော အင်္ဂါရုပ်တစ်ခုကို အကောင်အထည်ဖော်ခြင်း ပြုလုပ်သည်ဖြစ်စေ၊ ပါဝင်ကူညီသူများ (contributors) ထက် အသုံးပြုသူများ (users) က အဆပေါင်းများစွာ ပိုမိုများပြားပြီး ထိန်းသိမ်းသူများ (maintainers) ထက် ပါဝင် ကူညီသူများက အဆပေါင်းများစွာ ပိုမိုများပြားသည်ကို သတိရရန် ထိုက်တန်ပါတယ်။ ရလဒ်အနေဖြင့် ထိန်းသိမ်းသူများ၏ အချိန်သည် အလွန်ပင် မလောက်မငှ ဖြစ်နေရပါတယ်။ အကယ်၍ သင့် ပါဝင်ကူညီ မှုသည် အကျိုးရှိသော နေရာသို့ ရောက်ရှိနိုင်ခြေ တိုးတက်စေချင်ပါက သင်၏ ပါဝင်ကူညီမှုများသည် သတင်းအချက်အလက် ခိုင်မာမှု (signal-to-noise ratio) မြင့်မားပြီး ထိန်းသိမ်းသူများ၏ အချိန်နှင့် ထိုက်တန် ကြောင်း သေချာစေရပါမည်။

ဥပမာအားဖြင့်၊ ကောင်းမွန်သော Bug အစီရင်ခံစာသည် ပြဿနာကို နားလည်ရန်နှင့် ပြန်လည်ဖန်တီး စမ်းသပ် ရန် (reproduce) လိုအပ်သော အရာအားလုံးကို ပံ့ပိုးပေးခြင်းဖြင့် ထိန်းသိမ်းသူ၏ အချိန်ကို လေးစားမှု ပြသပါ တယ် -

- **ပတ်ဝန်းကျင် (Environment)**: OS၊ ဗားရှင်းနံပါတ်များ၊ သက်ဆိုင်ရာ ဆက်တင် ကွန်ဖစ်ဂူရေးရှင်းများ
- **သင် မျှော်လင့်ထားသည့် အရာ** နှင့် **အမှန်တကယ် ဖြစ်ပျက်ခဲ့သည့် အရာ**
- **ပြန်လည်ဖန်တီးရန် အဆင့်များ (Steps to reproduce)**: တိကျပါစေ။ “ခလုတ်ကို နှိပ်ပါ” ဟု ရေးခြင်း သည် “Admin အဖြစ် ဝင်ရောက်ထားစဉ် /settings စာမျက်နှာရှိ Submit ခလုတ်ကို နှိပ်ပါ” ဟု ရေးခြင်း လောက် အသုံးမဝင်ပါ။
- **သင် စမ်းသပ် ပြုလုပ်ခဲ့ပြီးသော အရာများ**: ၎င်းသည် ထပ်မံ အကြံပြုချက်များကို တားဆီးပေးပြီး သင် ကိုယ်တိုင် အချို့သော စုံစမ်းစစ်ဆေးမှုများ ပြုလုပ်ခဲ့ကြောင်း ပြသပါတယ်။

> အကယ်၍ သင်သည် လုံခြုံရေးဆိုင်ရာ အားနည်းချက်တစ်ခုကို တွေ့ရှိပါက၊ ၎င်းကို အများပြည်သူသို့ လျှို့ဝှက်ချက်မထားဘဲ မတင်ပါနှင့်။ မထုတ်ပြန်မီ ပုဂ္ဂလိကအဖြစ် ထိန်းသိမ်းသူများထံ စတင် ဆက်သွယ်ပြီး ပြင်ဆင်ရန် သင့်တော်သော အချိန် ပေးပါ။ ပရောဂျက်အများအပြားတွင် ဤရည်ရွယ်ချက်အတွက် `SECURITY.md` ဖိုင် သို့မဟုတ် ယင်းနှင့် ဆင်တူသော ဖိုင်များ ရှိကြပါတယ်။

ရှိပြီးသား ပြဿနာများ (issues) ကို ရှာဖွေထားကြောင်း သေချာပါစေ။ သင်၏ Bug သို့မဟုတ် အင်္ဂါရုပ်တောင်းဆိုချက်သည် တင်ပြပြီး ဖြစ်နိုင်ပြီး၊ ထပ်တူ ပြဿနာများ ထပ်မံ ဖန်တီးမည့်အစား ရှိပြီးသား ဆွေးနွေးမှုများတွင် သတင်းအချက်အလက် ထပ်မံဖြည့်စွက်ခြင်းက မဆိုင်းမတွ ပိုမို ကောင်းမွန်ပါတယ်။ ထို့အပြင် ၎င်းသည် ထိန်းသိမ်းသူများအတွက် အနှောင့်အယှက်များကိုလည်း လျှော့ချပေးပါတယ်။

အနည်းဆုံး ပြန်လည်ဖန်တီးနိုင်သော နမူနာများ (Minimal reproducible examples) ကို သင် ဖန်တီးပေးနိုင်ပါက အလွန်ပင် တန်ဖိုးရှိပါတယ်။ ၎င်းတို့သည် ထိန်းသိမ်းသူ၏ အချိန်နှင့် အားထုတ်မှုကို များစွာ သက်သာစေပြီး၊ Bug ကို စိတ်ချယုံကြည်စွာ ပြန်လည်ဖန်တီးခြင်းသည် ပြုပြင်ရန် အခက်ခဲဆုံး အပိုင်းဖြစ်လေ့ရှိပါတယ်။ ထို့အပြင် ပြဿနာကို သီးခြားခွဲထုတ်ရန် သင် စိုက်ထုတ်ခဲ့သော ကြိုးပမ်းမှုသည် ၎င်းကို ပိုမို နားလည်စေရန် ကူညီပေးပြီး အချို့သောအခါများတွင် ဖြေရှင်းချက်ကို မိမိကိုယ်တိုင် တွေ့ရှိသွားစေနိုင်ပါတယ်။

အကယ်၍ သင် ချက်ချင်း အကြောင်းပြန်စာ မရရှိပါက၊ ထိန်းသိမ်းသူများသည် ကန့်သတ်ထားသော အချိန်သာရှိသည့် စေတနာ့ဝန်ထမ်းများ ဖြစ်လေ့ရှိသည်ကို သတိရပါ။ အကယ်၍ သူတို့ထံမှ အကြောင်းပြန်စာကို စောင့်ဆိုင်းနေပါက သီတင်းပတ်အနည်းငယ် ကြာပြီးနောက် ယဉ်ကျေးစွာ ထပ်မံ မေးမြန်းခြင်းက အဆင်ပြေသော်လည်း နေ့စဉ် တွန်းအားပေး မေးမြန်းခြင်းမျိုး မပြုလုပ်သင့်ပါ။ ထို့အတူ “ကျွန်ုပ်လည်း ဖြစ်သည်” (“me too”) ဟူသော ကွန်မန့်များ သို့မဟုတ် Terminal output များကို Copy-Paste မျှသာ လုပ်ထားသော Bug အစီရင်ခံစာများသည် သင်၏ ပြဿနာ တိုးတက်မှု ရရှိရန်အတွက် အနုတ်လက္ခဏာ ဆောင်လေ့ရှိပါတယ်။

အကယ်၍ သင်သည် ကုဒ်ဖြင့် ပါဝင်ကူညီရန် မျှော်လင့်ပါက ပါဝင်ကူညီမှုဆိုင်ရာ လမ်းညွှန်ချက်များနှင့် ရင်းနှီးကျွမ်းဝင်အောင် ပြုလုပ်လိုပါလိမ့်မည်။ ပရောဂျက် အများအပြားတွင် `CONTRIBUTING.md` ဖိုင် ရှိကြပါတယ် - ၎င်းကို လိုက်နာပါ။ ထို့အပြင် သေးငယ်သောအရာမှ စတင်လိုပါလိမ့်မည် - စာလုံးပေါင်း အမှားပြုပြင်ခြင်း သို့မဟုတ် Documentation တိုးတက်အောင် ပြုလုပ်ခြင်းသည် ပရောဂျက်၏ လုပ်ငန်းစဉ်များကို သင်ယူနိုင်စေသောကြောင့် ပထမဆုံး ပါဝင်ကူညီမှုအဖြစ် အလွန်ကောင်းမွန်ပြီး၊ အကြောင်းအရာဆိုင်ရာ အပြင်းအထန် အပြန်အလှန် ဆွေးနွေးမှုများကို ဖြတ်သန်းရန် မလိုအပ်ပါ။

> ပရောဂျက် အသုံးပြုထားသော လိုင်စင်ကို စစ်ဆေးပါ၊ အကြောင်းမှာ သင့် ပါဝင်ကူညီသည့် ကုဒ်များသည်လည်း ထိုလိုင်စင်အောက်သို့ ရောက်ရှိမည်ဖြစ်သောကြောင့် ဖြစ်ပါတယ်။ အထူးသဖြင့် စေတနာလိုင်စင်များ (Copyleft licenses၊ ဥပမာ - GPL) ကို သတိပြုပါ၊ ၎င်းသည် ဆင်းသက်လာသော ကုဒ်များကိုလည်း အိုးပင်းဆော့စ်အဖြစ် သတ်မှတ်ရန် လိုအပ်ပြီး သင် ကိုင်တွယ်ပါက သင့် အလုပ်ရှင်အတွက် ရိုက်ခတ်မှုများ ရှိလာနိုင်ပါတယ်! **choosealicense.com** (https://choosealicense.com/) တွင် ပိုမို အသုံးဝင်သော သတင်းအချက်အလက်များ ရှိပါတယ်။

Pull request (“PR”) တစ်ခု စတင်ရန် ဆုံးဖြတ်ပြီးပါက၊ ပထမဦးစွာ အမှန်တကယ် လက်ခံစေချင်သော ပြောင်းလဲမှုကို သီးခြားခွဲထုတ်ထားကြောင်း သေချာပါစေ။ အကယ်၍ သင့် PR သည် အခြား ဆက်စပ်မှုမရှိသော အရာများစွာကို တစ်ပြိုင်နက်တည်း ပြောင်းလဲပါက သုံးသပ်သူ (reviewer) က ရှင်းလင်းရန် တောင်းဆိုပြီး သင့်ထံ ပြန်လည် ပို့ဆောင်ပေးနိုင်ခြေ ရှိပါတယ်။ ဤသည်မှာ git commits များကို ဆက်စပ်မှုရှိသော အစိတ်အပိုင်းများအဖြစ် ခွဲခြားသင့်သည့် နည်းလမ်းနှင့် ဆင်တူပါတယ်။

အချို့သော ကိစ္စများတွင် အကယ်၍ သင့်ထံ၌ သီးခြားစီဖြစ်နေပုံရသော ပြောင်းလဲမှုများစွာ ရှိသော်လည်း ၎င်းတို့ အားလုံးသည် အင်္ဂါရုပ်တစ်ခုအတွက် လိုအပ်ပါက ပြောင်းလဲမှု အားလုံး ပါဝင်သော ပိုမိုကြီးမားသည့် PR တစ်ခုကို ဖွင့်လှစ်ခြင်းက အဆင်ပြေနိုင်ပါတယ်။ သို့သော် ဤကိစ္စတွင် ထိန်းသိမ်းသူများအနေဖြင့် ပြောင်းလဲမှုကို “commit တစ်ခုချင်းစီအလိုက်” သုံးသပ်နိုင်သည့် အခွင့်အရေး ရရှိရန် commit သန့်ရှင်းမှုက အထူးပင် အရေးကြီးပါတယ်။

နောက်တစ်ခုအနေဖြင့်၊ ပြောင်းလဲမှု၏ နောက်ကွယ်မှ “အဘယ်ကြောင့်” (*why*) ကို သေချာစွာ ရှင်းပြပါ။ *ဘာတွေ* ပြောင်းလဲသွားသလဲဆိုသည်ကို ဖော်ပြရုံမျှ မပြုပါနှင့် - *အဘယ်ကြောင့်* ဤပြောင်းလဲမှု လိုအပ်သနည်းနှင့် *အဘယ်ကြောင့်* ဤနည်းလမ်းက ပြဿနာကို ဖြေရှင်းရန် ကောင်းမွန်သော နည်းလမ်းဖြစ်သနည်းဆိုသည်ကို ရှင်းပြပါ။ ပြောင်းလဲမှုတွင် အထူးသတိပြုရန် လိုအပ်သော အပိုင်းများရှိပါကလည်း ကြိုတင်၍ သတိပေးဖော်ပြသင့်ပါတယ်။ `CONTRIBUTING.md` နှင့် သင့်ပြောင်းလဲမှု၏ သဘောသဘာဝပေါ်မူတည်၍ သုံးသပ်သူများသည် ပြုလုပ်ခဲ့သော ရင်းနှီးပေးဆပ်ရမှုများ သို့မဟုတ် ပြောင်းလဲမှုကို မည်သို့ စမ်းသပ်ရမည်ဆိုသည့် အပိုဆောင်း သတင်းအချက်အလက်များကိုလည်း မျှော်လင့်နိုင်ပါတယ်။

> အနည်းဆုံး ပထမဆုံး နည်းလမ်းအဖြစ် ပရောဂျက်ကို “fork” လုပ်မည့်အစား မူရင်း Upstream ပရောဂျက်သို့ ပြန်လည် ပါဝင်ကူညီရန် အကြံပြုပါတယ်။ Fork လုပ်ခြင်းကို (လိုင်စင် ခွင့်ပြုပါက) သင် ပြုလုပ်လိုသော ပါဝင်ကူညီမှုများသည် မူရင်းပရောဂျက်၏ နယ်ပယ်ပြင်ပသို့ ရောက်ရှိနေသည့် အခါမှသာ အသုံးပြုသင့်ပါတယ်။ အကယ်၍ သင် Fork လုပ်ပါက မူရင်းပရောဂျက်ကို အသိအမှတ်ပြုထားကြောင်း သေချာပါစေ။

AI သည် လက်ခံနိုင်ဖွယ်ရှိသော ကုဒ်များနှင့် PR များကို လျင်မြန်စွာ ဖန်တီးရန် အလွန် လွယ်ကူစေသော်လည်း၊ ဤသည်မှာ သင် ပါဝင်ကူညီနေသော အရာကို နားမလည်ဘဲ နေခွင့် ပေးသည် မဟုတ်ပါ။ သင့်ကိုယ်တိုင် မရှင်းပြနိုင်သော AI ဖန်တီးထားသည့် ကုဒ်များကို တင်ပြခြင်းသည် ကုဒ်ရေးသားသူကိုယ်တိုင် မနားလည်သော ကုဒ်များကို စစ်ဆေးရန်နှင့် ထိန်းသိမ်းရန် ထိန်းသိမ်းသူများအပေါ် ဝန်ထုပ်ဝန်ပိုး ဖြစ်စေပါတယ်။ ပြဿနာများကို ရှာဖွေရန်နှင့် ပြုပြင်မှုများ/အင်္ဂါရုပ်များကို ထုတ်လုပ်ရန် AI အကူအညီ ရယူခြင်းက အဆင်ပြေပါတယ်၊ **သို့သော် ထိုအလုပ်ကို (ဝန်ပိနေပြီးဖြစ်သော) ထိန်းသိမ်းသူများထံ လွှဲပြောင်းပေးမည့် အစား တန်ဖိုးရှိသော ပါဝင်ကူညီမှုတစ်ခုဖြစ်အောင် သေချာစွာ စိစစ်ပြင်ဆင်ရန် မိမိတွင် တာဝန်ရှိပါတယ်။**

ထိန်းသိမ်းသူများအတွက် PR တစ်ခုကို လက်ခံခြင်းသည် ရေရှည် တာဝန်ယူမှုကို လက်ခံခြင်းဖြစ်သည်ကို သတိရပါ။ ပါဝင်ကူညီသူ ထွက်ခွာသွားပြီးနောက် အချိန်ကြာမြင့်စွာ ဤကုဒ်ကို သူတို့ ဆက်လက် ထိန်းသိမ်းရမည်ဖြစ်သဖြင့် စေတနာဖြင့် ပြုလုပ်ထားသော်လည်း ပရောဂျက်၏ ဦးတည်ချက်နှင့် မကိုက်ညီသော၊ သူတို့ မထိန်းသိမ်းချင်သော ရှုပ်ထွေးမှုများကို ပေါင်းစပ်ပေးသော၊ သို့မဟုတ် လိုအပ်ချက်ကို လုံလောက်စွာ မှတ်တမ်းမတင်ထားသော ပြောင်းလဲမှုများကို ငြင်းပယ်နိုင်ပါတယ်။ ပါဝင်ကူညီမှုကို လက်ခံခြင်းသည် ထိန်းသိမ်းမှု ဝန်ထုပ်ဝန်ပိုးနှင့် ထိုက်တန်ကြောင်း အကြောင်းပြချက် ပေးရန်မှာ ပါဝင်ကူညီသူ သင့် အပေါ်တွင် တည်ရှိပါတယ်။

> PR ဆိုင်ရာ တုံ့ပြန်ချက်များကို လက်ခံရရှိသည့်အခါ သင့် ကုဒ်သည် သင့်ကိုယ်တိုင် မဟုတ်ကြောင်း သတိရပါ! သုံးသပ်သူများသည် ကုဒ်ကို ပိုမို ကောင်းမွန်အောင် ပြုလုပ်ရန် ကြိုးစားနေခြင်းဖြစ်ပြီး သင့်ကို ပုဂ္ဂိုလ်ရေးအရ ဝေဖန်နေခြင်း မဟုတ်ပါ။ အကယ်၍ သင် သဘောမတူပါက ရှင်းလင်းချက် မေးခွန်းများ မေးမြန်းပါ - သင် တစ်ခုခု သင်ယူရရှိနိုင်သလို၊ သို့မဟုတ် သူတို့လည်း သင်ယူရရှိနိုင်ပါတယ်။

ပြန်လည်သုံးသပ်ခြင်း (Reviewing)

Code review ကို Senior Developer များသာ ပြုလုပ်သည်ဟု သင် ထင်မြင်နိုင်သော်လည်း၊ သင် မျှော်လင့်ထားသည်ထက် ပိုမို စောစီးစွာ ကုဒ် သုံးသပ်ပေးရန် တောင်းဆိုခံရနိုင်ပြီး သင့် အမြင်သည်လည်း တန်ဖိုးရှိပါတယ်။ အသစ်အဆန်း အမြင်များသည် အတွေ့အကြုံရှိ Developer များ သတိမထားမိသော အရာများကို တွေ့ရှိနိုင်ပြီး၊ ကုဒ်နှင့် ရင်းနှီးမှုနည်းသူတစ်ဦး၏ မေးခွန်းများသည် မကြာခဏဆိုသလို မှတ်တမ်းတင်ရမည့် သို့မဟုတ် ရိုးရှင်းအောင် ပြုလုပ်ရမည့် ယူဆချက်များကို ပေါ်လွင်စေပါတယ်။

Review ပြုလုပ်ခြင်းသည် လေ့လာရန် အမြန်ဆုံး နည်းလမ်းများအနက် တစ်ခုလည်း ဖြစ်ပါတယ်။ အခြားသူများ ပြဿနာများကို မည်သို့ ချဉ်းကပ်သနည်းဆိုသည်ကို တွေ့မြင်ရမည်ဖြစ်ပြီး၊ ဒီဇိုင်း ပုံစံများ (patterns) နှင့် အသုံးအနှုန်းများ (idioms) ကို လေ့လာနိုင်ကာ မည်သည့်အရာက ကုဒ်ကို ဖတ်ရှုရလွယ်ကူစေသနည်းဆို

သည့် စာနာနားလည်မှုကို တည်ဆောက်နိုင်မည် ဖြစ်ပါတယ်။ ကိုယ်ပိုင် တိုးတက်မှုအပြင်၊ Review များသည် Production သို့ မရောက်မီ Bug များကို ဖမ်းဆီးပေးနိုင်ခြင်း၊ အဖွဲ့တစ်လျှောက် အသိပညာ ပြန့်ပွားစေခြင်း၊ နှင့် ပူးပေါင်းဆောင်ရွက်မှုမှတစ်ဆင့် ကုဒ်အရည်အသွေးကို တိုးတက်စေခြင်းတို့ကို ဆောင်ရွက်ပေးပါတယ်။ ၎င်းတို့သည် ရုံးလုပ်ငန်းဆိုင်ရာ ဝန်ထုပ်ဝန်ပိုး သက်သက် မဟုတ်ပါဘူး။

ကောင်းမွန်သော ကုဒ် သုံးသပ်ခြင်းသည် အချိန်ယူ၍ လေ့ကျင့်ယူရမည့် ကျွမ်းကျင်မှုတစ်ခု ဖြစ်သော်လည်း၊ ၎င်းတို့ကို ပိုမို မြန်ဆန်စွာ ကောင်းမွန်စေနိုင်သော အကြံပြုချက်အချို့ ရှိပါတယ် -

- **လူကို မဟုတ်ဘဲ ကုဒ်ကို သုံးသပ်ပါ**: “ဤ Function သည် ရှုပ်ထွေးသည်” နှင့် “သင် ရှုပ်ထွေးသော ကုဒ် ရေးထားသည်” ကို နှိုင်းယှဉ်ကြည့်ပါ။
- **အမှန်တကယ် ဆောင်ရွက်နိုင်သော ကွန်မန့်များကို ပိုမို သုံးပါ**: “ဤ globals များကို config dataclass ဖြင့် အစားထိုးနိုင်မလား” ဟူသော ကွန်မန့်သည် “ဒီမှာ globals မသုံးပါနှင့်” ထက် ဖြေရှင်းရန် ပိုမို လွယ်ကူပါတယ်။
- **တောင်းဆိုချက်များ ပြုလုပ်မည့်အစား မေးခွန်းများ မေးပါ**: “ဒီမှာ X က null ဖြစ်သွားရင် ဘာဖြစ်မလဲ” ဟု မေးခြင်းက “Null ဖြစ်စဉ်ကို ကိုင်တွယ်ပါ” ဟု ပြောခြင်းထက် ဆွေးနွေးမှုကို ပိုမို ဖိတ်ခေါ်ပါတယ်။
- **“အဘယ်ကြောင့်” ကို ရှင်းပြပါ**: “ဒီမှာ ကိန်းသေ (constant) သုံးရန် စဉ်းစားပါ” ဟု ပြောခြင်းသည် “ပတ်ဝန်းကျင်အလိုက် Timeout ကို လွယ်ကူစွာ ပြင်ဆင်နိုင်ရန် ဒီမှာ ကိန်းသေ (constant) သုံးရန် စဉ်းစားပါ” ထက် အသုံးဝင်မှု နည်းပါတယ်။
- **တားဆီးသည့် ပြဿနာများနှင့် အကြံပြုချက်များကို ခွဲခြားပါ**: မည်သည့်အရာ မဖြစ်မနေ ပြောင်းလဲရမည်နှင့် မည်သည့်အရာက စိတ်ကြိုက် ရွေးချယ်မှုဖြစ်သည်ကို ရှင်းလင်းစွာ ဖော်ပြပါ။
- **ကောင်းမွန်သော အရာများကို အသိအမှတ်ပြုပါ**: ပါးနပ်သော ဖြေရှင်းချက်များ သို့မဟုတ် သန့်ရှင်းသော ရေးသားမှုများကို ထောက်ပြခြင်းသည် အားပေးရာ ရောက်ပြီး ရေးသားသူအား မည်သည့်အရာကို ဆက်လက် ပြုလုပ်ရမည်ကို သိရှိစေပါတယ်။
- **မည်သည့်အခါ ရပ်တန့်ရမည်ကို သိရှိပါ**: ပါဝင်ကူညီသူများတွင် အချိန်နှင့် စိတ်ရှည်မှု အကန့်အသတ်ရှိပြီး၊ အသေးအဖွဲ ကိစ္စရပ်များအားလုံးကို ကိုင်တွယ်ခြင်းက အကောင်းဆုံး မဟုတ်ပါဘူး။ အဓိက အရာများကို အာရုံစိုက်ပါ၊ ထို့နောက် အသေးအဖွဲများကို နောင်တွင် သင်ကိုယ်တိုင် သန့်ရှင်းရေး ပြုလုပ်ရန် စဉ်းစားပါ။

> AI tools များသည် အချို့သော ပြဿနာများကို ဖမ်းဆီးနိုင်သော်လည်း လူသားများ၏ သုံးသပ်မှု အတွက် အစားထိုး မဟုတ်ပါဘူး။ ၎င်းတို့သည် အကြောင်းအရင်းများ (context) ကို လွတ်သွားတတ်ပြီး ပရောဂျက် လိုအပ်ချက်များကို မနားလည်ပါ၊ ထို့ပြင် လွဲမှားသော အရာများကို စိတ်ချလက်ချ အကြံပြုနိုင်ပါတယ်။ ၎င်းတို့ကို ပထမဆုံး အကြမ်းစစ်ဆေးမှုအဖြစ် အသုံးပြုရန် ထိုက်တန်သော်လည်း သေချာစွာ စဉ်းစားထားသော လူသား သုံးသပ်မှု၏ အစားထိုး မဟုတ်ပါဘူး။

လေ့လာသင်ယူခြင်းနှင့် မျှဝေခြင်း (Education)

အင်ဂျင်နီယာများအနေဖြင့် ကုဒ်မရေးသည့် အချိန်အများစုကို မေးခွန်းများ မေးခြင်း သို့မဟုတ် ဖြေကြားခြင်းတို့ဖြင့် ကုန်လွန်စေရပါတယ်၊ ပူးပေါင်းဆောင်ရွက်စဉ်၊ လုပ်ဖော်ကိုင်ဖက်များနှင့် ဆွေးနွေးစဉ် သို့မဟုတ် လေ့လာရန် ကြိုးစားနေစဉ်တွင် နှစ်ခုစလုံး ရောနှောပါဝင်နိုင်ပါတယ်။ မေးခွန်းကောင်းများ မေးမြန်းခြင်းသည် အထူးကျွမ်းကျင်စွာ ရှင်းပြနိုင်သူများထံမှ သာမက မည်သူ့ထံမှမဆို ပိုမို ကောင်းမွန်စွာ သင်ယူနိုင်စေသည့် ကျွမ်းကျင်မှုတစ်ခု ဖြစ်ပါတယ်။ Julia Evans ၏ “[မေးခွန်းကောင်းများ မည်သို့ မေးရမလဲ](https://jvns.ca/blog/good-questions/)” နှင့် “[သင့် မေးခွန်းများအတွက် အသုံးဝင်သော အဖြေများ မည်သို့ ရယူမလဲ](https://jvns.ca/blog/2021/10/21/how-to-get-useful-answers-to-your-questions/)” ဘလော့ဂ်ပို့စ်များသည် ဖတ်ရှုရန် အထူး ထိုက်တန်ပါတယ်။

အထူးတလွန် တန်ဖိုးရှိသော အကြံပြုချက်အချို့မှာ -

- **သင် နားလည်ထားသည်ကို ပထမဦးစွာ ဖော်ပြပါ**: သင် သိထားသည်ဟု ထင်ရသော အရာကို ပြောပြီး “အဲဒါ မှန်သလား” ဟု မေးပါ။ ဤသည်မှာ အဖြေပေးသူအား သင်၏ အမှန်တကယ် မသိသေးသော ကွက်လပ်များကို ဖော်ထုတ်ရန် ကူညီပေးပါတယ်။
- **ဟုတ်/မဟုတ် မေးခွန်းများ မေးပါ**: “X က မှန်ပါသလား” ဟူသော မေးခွန်းသည် မသက်ဆိုင်သော ရှင်းလင်းချက်များကို တားဆီးပေးပြီး အသုံးဝင်သော အသေးစိတ် ရှင်းလင်းချက်များကို ရရှိစေလေ့ရှိပါတယ်။
- **တိကျပါစေ**: “SQL joins များ မည်သို့ အလုပ်လုပ်သနည်း” ဟူသော မေးခွန်းသည် လွန်းမက ကျယ်ပြန့်ပါတယ်။ “LEFT JOIN တွင် ညာဘက် table ၌ ကိုက်ညီမှုမရှိသော rows များ ပါဝင်ပါသလား” ဟူသော မေးခွန်းမျိုးက အဖြေပေးနိုင်ပါတယ်။
- **မနားလည်ပါက ဝန်ခံပါ**: မရင်းနှီးသော အသုံးအနှုန်းများကို မေးရန် ကြောဖြတ် မေးမြန်းပါ။ ဤသည်မှာ ယုံကြည်မှုကို ပြသခြင်းဖြစ်ပြီး အားနည်းချက် မဟုတ်ပါ။ ထို့အတူ အကယ်၍ သူတို့က သင့်အား သင်မသိသော မေးခွန်းများ မေးပါက “ကျွန်ုပ် မသိပါ” ဟု ပြောခြင်းက အကောင်းဆုံးဖြစ်ပြီး၊ “သို့သော် ကျွန်ုပ် ထင်သည်မှာ...” သို့မဟုတ် “သို့သော် ကျွန်ုပ် စုံစမ်းကြည့်နိုင်ပါတယ်” ဟု ဆက်လက် ပြောဆိုနိုင်ပါတယ်။
- **မပြည့်စုံသော အဖြေများကို လက်မခံပါနှင့်**: သင် အမှန်တကယ် နားလည်သည်အထိ ဆက်လက်၍ မေးခွန်းများ မေးပါ။
- **ရှာဖွေ လေ့လာမှုအချို့ ပြုလုပ်ပါ**: အခြေခံ စုံစမ်းစစ်ဆေးမှုသည် ပိုမို တိကျသော မေးခွန်းများ မေးမြန်းနိုင်ရန် ကူညီပေးပါတယ် (လုပ်ဖော်ကိုင်ဖက်များအကြား ပေါ့ပေါ့ပါးပါး မေးခွန်းများမှာမူ အဆင်ပြေပါတယ်)။

သတိရပါ - သေချာစွာ စဉ်းစားထားသော မေးခွန်းများသည် အသိုင်းအဝိုင်းတစ်ခုလုံးကို အကျိုးပြုပါတယ်။ ၎င်းတို့သည် အခြားသူများလည်း နားလည်ရန် လိုအပ်သော ကွယ်ဝှက်နေသည့် ယူဆချက်များကို ပေါ်လွင်စေပါတယ်။

> ဤအကြံပြုချက်သည် LLM များနှင့် ဆက်သွယ်ပြောဆိုသည့်အခါတွင်လည်း ထို့အတူ သက်ရောက်ကြောင်း သတိပြုပါ!

AI အသုံးပြုမှု ကျင့်ဝတ်များ (AI etiquette)

ဆော့ဖ်ဝဲ အင်ဂျင်နီယာ ရပ်ဝန်းတွင် LLM များနှင့် AI များကို အသုံးပြုမှု တိုးတက်လာသည်နှင့်အမျှ၊ ယင်းတို့နှင့် ပတ်သက်သည့် လူမှုရေးနှင့် အသက်မွေးဝမ်းကျောင်းဆိုင်ရာ စံနှုန်းများသည် ပြောင်းလဲနေဆဲ ဖြစ်ပါတယ်။ ကျွန်ုပ်တို့သည် နည်းဗျူဟာမြောက် စဉ်းစားဖွယ်ရာ အများအပြားကို [Agentic Coding ခေါင်းစဉ်](https://missing-semester-my.github.io/2026/agentic-coding/) တွင် လွှမ်းခြုံခဲ့ပြီး ဖြစ်သော်လည်း ၎င်းတို့ အသုံးပြုမှု၏ အခြားသော "နူးညံ့သည့်" (softer) အပိုင်းများကိုလည်း ဆွေးနွေးရန် ထိုက်တန်ပါတယ်။

ပထမဆုံးအချက်မှာ အကယ်၍ AI သည် သင်၏ လုပ်ငန်းတွင် အဓိပ္ပာယ်ရှိစွာ ပါဝင်ကူညီခဲ့ပါက **ပွင့်လင်းစွာ ထုတ်ဖော်ပြောဆိုပါ**။ ဤသည်မှာ ရှက်စရာ မဟုတ်ပါ - ရိုးသားမှု၊ သင့်တော်သော မျှော်လင့်ချက်များ သတ်မှတ်မှု၊ နှင့် ထွက်ပေါ်လာသော အလုပ်သည် သင့်တော်သော Review အဆင့် ရရှိစေရေးတို့အတွက် ဖြစ်ပါတယ်။ မည်သည့် *အပိုင်းများ* အတွက် AI ကို အသုံးပြုခဲ့သည်ကို ထုတ်ဖော်ပြောဆိုခြင်းကလည်း တန်ဖိုးရှိပါတယ် - "ဤအရာတစ်ခုလုံးကို vibecode လုပ်ထားခြင်းဖြစ်သည်" နှင့် "ကျွန်ုပ်သည် ဤ Backup tool ကို ရေးသားခဲ့ပြီး Web frontend ဒီဇိုင်းအတွက် LLM ကို အသုံးပြုခဲ့သည်" ဟူသည်အကြား ထင်ရှားသော ခွဲခြားမှု ရှိပါတယ်။ ဥပမာအားဖြင့်၊ ကျွန်ုပ်တို့သည် စာလုံးပေါင်း စီစစ်ခြင်း၊ အကြံဉာဏ် ဖလှယ်ခြင်း၊ နှင့် ကုဒ်နမူနာများနှင့် လေ့ကျင့်ခန်းများ၏ ပထမဆုံး မူကြမ်းများ ထုတ်လုပ်ခြင်းတို့ အပါအဝင် ဤ သင်ခန်းစာ မှတ်စုအချို့ကို ရေးသားရာတွင် ကူညီရန် LLM များကို အသုံးပြုခဲ့ပါတယ်။

သင် ပါဝင်ကူညီနေသော အဖွဲ့များနှင့် ပရောဂျက်များ၏ စံနှုန်းများကိုလည်း လိုက်နာလိုပါလိမ့်မည်။ အချို့ အဖွဲ့များတွင် အခြားအဖွဲ့များထက် AI အသုံးပြုမှုဆိုင်ရာ ပိုမို တင်းကျပ်သော မူဝါဒများ ရှိကြပါတယ် (ဥပမာ - လိုက်နာဆောင်ရွက်မှု သို့မဟုတ် ဒေတာ တည်ရှိမှု အကြောင်းအရင်းများ ကြောင့်ဖြစ်သည်)၊ ထို့ကြောင့် မတော်တဆ စည်းကမ်းဖောက်ဖျက်မိခြင်းမျိုး မဖြစ်ချင်ပါ။ သင်၏ အသုံးပြုမှုကို ပွင့်လင်းစွာ ဖော်ပြခြင်းသည် ကုန်ကျစရိတ် ကြီးမားနိုင်သော အမှားများကို တားဆီးရန် ကူညီပေးပါတယ်။

> အကယ်၍ သင်သည် ပြုလုပ်နေသော အလုပ်မှတစ်ဆင့် လေ့လာရန် ရည်ရွယ်ပါက၊ AI အား အလုပ်အားလုံး သို့မဟုတ် အများစုကို ခိုင်းစေခြင်းသည် မိမိကိုယ်တိုင် တိုးတက်မှုကို အဟန့်အတား ဖြစ်စေနိုင်ကြောင်း သတိရပါ - သင်သည် လုပ်ငန်းစဉ်ကိုယ်တိုင်ထက် Prompting ရေးသားခြင်း (နှင့် AI ရလဒ်များကို Review ပြုလုပ်ခြင်း) အကြောင်းကိုသာ ပိုမို လေ့လာမိပါလိမ့်မည်။ အထူးသဖြင့် သင် လေ့လာနေချိန်တွင် ပန်းတိုင်ထက် ခရီးစဉ် (ခရီးလမ်း) က အဓိက ဖြစ်နိုင်သည်၊ ထို့ကြောင့် "ဖြေရှင်းချက်ကို မြန်ဆန်စွာ ရရှိရန်" AI ကို အသုံးပြုခြင်းသည် ဆန့်ကျင်ဘက် ရည်မှန်းချက် (anti-goal) ဖြစ်ပါတယ်။

ဆက်စပ်နေသည့် စိုးရိမ်ဖွယ်ရာ တစ်ခုမှာ အင်တာဗျူးများနှင့် အခြားသော အကဲဖြတ်မှု အခြေအနေများတွင် ပေါ်ပေါက်လာပါတယ်။ ဤအရာများသည် LLM ၏ ကျွမ်းကျင်မှု မဟုတ်ဘဲ သင့် ကျွမ်းကျင်မှုနှင့် စွမ်းဆောင်ရည်များကို သီးသန့် အကဲဖြတ်ရန် ရည်ရွယ်လေ့ရှိပါတယ်။ ကုမ္ပဏီ အများအပြားသည် အင်တာဗျူး၏ အစိတ်အပိုင်းအဖြစ် ထို ဆက်သွယ်ဆောင်ရွက်မှုများကို လေ့လာခွင့်ပေးထားသရွေ့ အင်တာဗျူးများတွင် LLM များနှင့် အခြား AI အထောက်အကူပြု tools များကို အသုံးပြုခွင့် ပေးထားကြပြီး (ဆိုလိုသည်မှာ ထို tool များကို အသုံးပြုနိုင်သော သင့် ကျွမ်းကျင်မှုကိုပါ အကဲဖြတ်နေခြင်း ဖြစ်သည်!)၊ သို့သော် ထိုသို့သော ကုမ္ပဏီများသည် အနည်းစုသာ ရှိပါသေးတယ်။ အကယ်၍ သီးခြား လုပ်ငန်းတစ်ခုအတွက် AI အကူအညီ ရယူခွင့် ရှိမရှိ မသေချာပါက မေးမြန်းပါ!

> အကယ်၍ အကဲဖြတ်မှု အခြေအနေတစ်ခုက ပြင်ပ tool များ၊ LLMs များ စသည်တို့ကို မသုံးရဟု အတိအလင်း သတ်မှတ်ထားပါက ၎င်းတို့ကို မသုံးသင့်ကြောင်း သီးသန့် ပြောရန်ပင် မလိုပါဘူး။ မမိအောင် တိတ်တဆိတ် သုံးစွဲရန် ကြိုးစားခြင်းသည် သင့်ထံ အကျိုးဆက်အဖြစ် အမှန်တကယ် ပြန်လည် **ရိုက်ခတ်လာပါလိမ့်မည်**။

လေ့ကျင့်ခန်းများ (Exercises)

1. ထင်ရှားသော ပရောဂျက်တစ်ခု၏ စောစ်ကုဒ် (source code) ကို ဝင်ရောက်ကြည့်ရှုပါ (ဥပမာ - **Redis** (https://github.com/redis/redis) သို့မဟုတ် **curl** (https://github.com/curl/curl))။ သင်ခန်းစာတွင် ဖော်ပြထားသော ကွန်မန့် အမျိုးအစား အချို့၏ နမူနာများကို ရှာဖွေပါ - အသုံးဝင်သော TODO တစ်ခု၊ ပြင်ပ Documentation ဆိုင်ရာ ကိုးကားချက် တစ်ခု၊ ရှောင်ရှားခဲ့သော နည်းလမ်းကို ရှင်းပြထားသည့် “အဘယ်ကြောင့် မသုံးသနည်း” ကွန်မန့် တစ်ခု၊ သို့မဟုတ် ခက်ခဲစွာ သင်ယူခဲ့ရသော သင်ခန်းစာ ကွန်မန့် တစ်ခု။ အကယ်၍ ထို ကွန်မန့် မရှိပါက မည်သည့်အရာများ ဆုံးရှုံးသွားမည်နည်း။

2. သင် စိတ်ဝင်စားသော အိုးပင်းဆော့စ် ပရောဂျက်တစ်ခုကို ရွေးချယ်ပြီး ၎င်း၏ လတ်တလော commit သမိုင်းကြောင်း (`git log`) ကို ကြည့်ရှုပါ။ အဘယ်ကြောင့် ပြောင်းလဲမှုကို ပြုလုပ်ခဲ့သနည်း (*why*) ဆိုသည်ကို ရှင်းပြထားသော ကောင်းမွန်သည့် မက်ဆေ့ဂျ် ပါဝင်သည့် commit တစ်ခုနှင့် ဘာတွေ ပြောင်းလဲသွားသလဲ (*what*) ဆိုသည်ကိုသာ ဖော်ပြထားသည့် အားနည်းသော မက်ဆေ့ဂျ် ပါဝင်သည့် commit တစ်ခုကို ရှာပါ။ အားနည်းသော commit အတွက် diff ကို ကြည့်ရှုပြီး (`git show <hash>`) ပြဿနာ → ဖြေရှင်းချက် → နောက်ဆက်တွဲ ရလဒ်များ အဆင်လိုက် တည်ဆောက်ပုံအတိုင်း ပိုမို ကောင်းမွန်သော commit မက်ဆေ့ဂျ် ရေးသားရန် ကြိုးစားကြည့်ပါ။ ပြီးစီးသွားပြီးနောက် လိုအပ်သော အကြောင်းအရင်းများ (context) ကို ပြန်လည် စုစည်းရန် မည်မျှ ကြိုးစားအားထုတ်ရသည်ကို သတိပြုပါ!

3. Star ၁၀၀၀ ကျော် ရရှိထားသော GitHub ပရောဂျက် သုံးခု၏ README များကို နှိုင်းယှဉ်ကြည့်ပါ။ ၎င်းတို့ အားလုံးသည် တူညီစွာ အသုံးဝင်ကြပါသလား။ သင်ကိုယ်တိုင် နောင်တွင် ရေးသားမည့် README များအတွက် သင်ခန်းစာအဖြစ် သင့်အတွက် အနှောင့်အယှက် သက်သက်သာ ဖြစ်စေသည့် အရာများကို ရှာဖွေပါ။

4. သင် အသုံးပြုသော ပရောဂျက်တစ်ခုတွင် ဖွင့်လှစ်ထားသော Issue တစ်ခုကို ရှာပါ (၎င်းတို့တွင် ရှိပါက “good first issue” သို့မဟုတ် “help wanted” လေဘယ်များကို စစ်ဆေးပါ)။ ၎င်း Issue ကို သင်ခန်းစာပါ စံနှုန်းများနှင့် နှိုင်းယှဉ် အကဲဖြတ်ပါ - ၎င်းသည် ထိန်းသိမ်းသူ၏ အချိန်ကို တန်ဖိုးထားပြီး Debug ပြုလုပ်ရန် လိုအပ်သော သတင်းအချက်အလက် အားလုံး ပါဝင်ပုံ ရပါသလား၊ သို့မဟုတ် ပင်မ ပြဿနာသို့ ရောက်ရှိရန် ထိန်းသိမ်းသူအနေဖြင့် တင်ပြသူထံ အကြိမ်ကြိမ် မေးခွန်းများ မေးမြန်းရန် လိုအပ်မည်ဟု သင် မျှော်လင့်ပါသလား။

5. သင် အသုံးပြုသော ဆော့ဖ်ဝဲတွင် ကြုံတွေ့ခဲ့ဖူးသော Bug တစ်ခုကို စဉ်းစားပါ (သို့မဟုတ် Issue Tracker တွင် တစ်ခု ရှာဖွေပါ)။ အနည်းဆုံး ပြန်လည်ဖန်တီးနိုင်သော နမူနာ (minimal reproducible example) တစ်ခု ဖန်တီးရန် လေ့ကျင့်ပါ - ပြဿနာကို ပြသနိုင်ဆဲ အသေးငယ်ဆုံး အခြေအနေတစ်ခု ရရှိသည်အထိ Bug

နှင့် မသက်ဆိုင်သော အရာအားလုံးကို ဖယ်ရှားပါ။ သင် ဖယ်ရှားခဲ့သည်များနှင့် အဘယ်ကြောင့် ဖယ်ရှားခဲ့သည်ကို ရေးသားပါ။

6. သင် ရင်းနှီးသော ပရောဂျက်တစ်ခုတွင် အဓိပ္ပာယ်ရှိသော Review ကွန်မန့်များ ပါဝင်သည့် Merge လုပ်ပြီးသား Pull Request တစ်ခုကို ရှာပါ (“LGTM” သက်သက် မဟုတ်ပါ)။ Review ကို ဖတ်ရှုပါ။ ကွန်မန့် အားလုံးသည် တူညီစွာ အကျိုးရှိပါသလား။ အကယ်၍ သင်သည် PR ရေးသားသူ ဖြစ်ပါက ထို ကွန်မန့်များ အအားလုံး ရရှိသည့် အတွေ့အကြုံကို မည်သို့ ခံစားရမည်နည်း။

7. Stack Overflow သို့ သွားရောက်ပြီး သင်သိသော နည်းပညာတစ်ခုတွင် မဲအများအပြား ရရှိထားသည့် အဖြေပါသော မေးခွန်းတစ်ခုကို ရှာပါ။ ထို့နောက် ပိတ်သိမ်းထားသော သို့မဟုတ် မဲလျော့ခြင်း အများအပြား ခံရသော မေးခွန်းတစ်ခုကို ရှာပါ။ ၎င်းတို့ကို သင်ခန်းစာပါ အကြံပြုချက်များနှင့် နှိုင်းယှဉ်ကြည့်ပါ - မည်သည့် မေးခွန်းက ပိုမို ကောင်းမွန်သော အဖြေများ ရရှိမည်ဆိုသည်ကို ကြိုတင် ခန့်မှန်းနိုင်ပါသလား။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

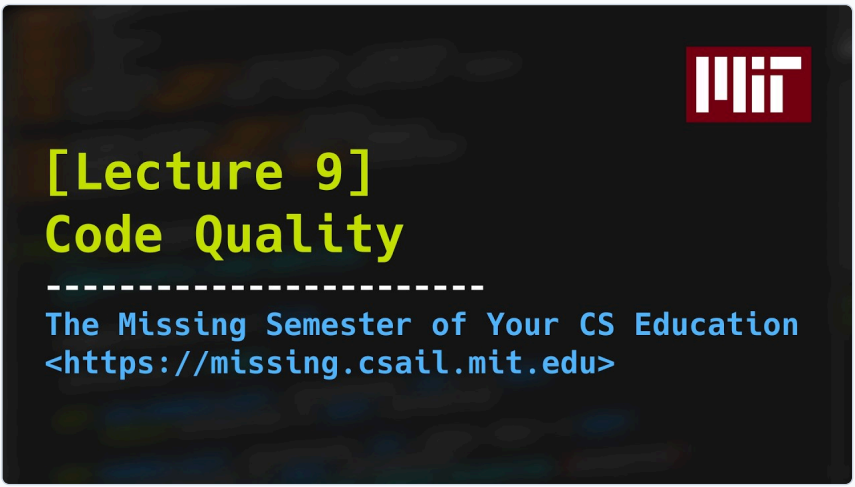
Resource / Description	URL	Micro QR
1. choosealicense.com	https://choosealicense.com/	
2. မေးခွန်းကောင်းများ မည်သို့ မေးရမလဲ	https://jvns.ca/blog/good-questions/	
3. သင့် မေးခွန်းများအတွက် အသုံးဝင်သော အဖြေများ မည်သို့ ရယူမလဲ	https://jvns.ca/blog/2021/10/21/how-to-get-useful-answers-to-your-questions/	
4. Agentic Coding ခေါင်းစဉ်	https://missing-semester-my.github.io/2026/agentic-coding/	
5. Redis	https://github.com/redis/redis	
6. curl	https://github.com/curl/curl	

Code Quality

အနှစ်ချုပ် - Formatting၊ linting၊ testing၊ continuous integration နှင့် အခြား အကြောင်းအရာများ အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Code Quality



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=XBiLUNx84CQ

Developer များ အရည်အသွေးမြင့် ကုဒ်များ ရေးသားနိုင်ရန် ကူညီပေးသည့် tool များနှင့် နည်းလမ်းများစွာ ရှိပါသည်။ ဤသင်ခန်းစာတွင် အောက်ပါတို့ အကြောင်းကို လွှမ်းခြုံ ဖော်ပြသွားပါမည် -

- **Formatting**
- **Linting**
- **Testing**
- **Pre-commit hooks**
- **Continuous integration**
- **Command runners**

အပိုဆောင်း ခေါင်းစဉ်တစ်ခု အနေဖြင့် **regular expressions** (regex) အကြောင်းကိုလည်း လွှမ်းခြုံ ဖော်ပြသွားပါမည်။ regex သည် ကုဒ် အရည်အသွေး ထိန်းသိမ်းရာတွင် (ဥပမာ - pattern တစ်ခုနှင့် ကိုက်ညီသော test အစိတ်အပိုင်းများကို ရွေးချယ် စိစစ် run ရာတွင်) သာမက IDE များကဲ့သို့ အခြားသော နယ်ပယ်များတွင်ပါ (ဥပမာ - ရှာဖွေပြီး အစားထိုးရာတွင်) ကျယ်ပြန့်စွာ အသုံးပြုနိုင်သည့် နည်းပညာတစ်ခု ဖြစ်ပါသည်။

ဤ tool အများစုသည် သက်ဆိုင်ရာ programming language အလိုက် သီးခြား ဖြစ်ကြပါလိမ့်မည် (ဥပမာ - Python အတွက် **Ruff** (https://docs.astral.sh/ruff/) linter/formatter)။ အချို့သော သာဓကများတွင်မူ tool များသည် language အများအပြားကို ပံ့ပိုးပေးနိုင်ပါသည် (ဥပမာ - **Prettier** (https://prettier.io/) code formatter)။ သို့သော်လည်း သဘောတရားများမှာမူ နေရာတိုင်းလိုလိုတွင် အတူတူပင် ဖြစ်ပါသည် — မည်သည့် programming language အတွက်မဆို code formatter များ၊ linter များ၊ testing library များနှင့် အခြား tool များကို ရှာဖွေ ရရှိနိုင်ပါသည်။

Formatting

Code auto-formatter များသည် စာသားဆိုင်ရာ syntax များ၏ ရှုပသွင်ပြင်ကို အလိုအလျောက် သပ်ရပ်လှပစေပါသည်။ ဤနည်းဖြင့် auto-formatting tool က string များအတွက် `'` သို့မဟုတ် `"` စာလုံး ပုံစံ ညီညွတ်မှု၊ binary operator များ၏ ဘေးတွင် space ခြားခြင်း (`x+y` အစား `x + y`)၊ `import` statement များကို အစဉ်လိုက် စီစဉ်ထားခြင်း နှင့် လိုင်းအရှည် လွန်ကဲမှုကို ရှောင်လွှဲခြင်း ကဲ့သို့သော အသေးစိတ် ကိစ္စရပ်များကို အလိုအလျောက် ကိုင်တွယ်ပေးနေစဉ် သင်သည် ပိုမို နက်နဲပြီး စိန်ခေါ်မှု ရှိသော ပြဿနာများကိုသာ အာရုံစိုက်နိုင်မည် ဖြစ်ပါသည်။ code formatter များ၏ အဓိက အကျိုးကျေးဇူး တစ်ခုမှာ codebase တစ်ခုပေါ်တွင် လုပ်ဆောင်နေကြသော Developer အားလုံး၏ ကုဒ်စတိုင်လ်ကို စံနှုန်းတစ်ခုတည်း ဖြစ်အောင် ညှိပေးနိုင်ခြင်း ဖြစ်ပါသည်။

Prettier ကဲ့သို့သော tool အချို့သည် [စိတ်ကြိုက် ပြင်ဆင်ရန် စက်ဝန်း ကျယ်ပြန့်ပါသည်](https://prettier.io/docs/configuration); မိမိ ပရောဂျက်၏ configuration file ကို **version control** (https://missing-semester-my.github.io/2026/version-control/) ထဲသို့ ရောက်အောင် check in ပြုလုပ်ထားသင့်ပါသည်။ **Black** (https://github.com/psf/black) နှင့် **gofmt** (https://pkg.go.dev/cmd/gofmt) ကဲ့သို့သော အခြား tool များတွင်မူ မလိုအပ်ဘဲ အသေးအဖွဲ ကိစ္စများဖြင့် ငြင်းခုံနေခြင်း (**bikeshedding** (https://en.wikipedia.org/wiki/Law_of_triviality)) ကို လျှော့ချရန်အတွက် စိတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်းကို အကန့်အသတ်ဖြင့်သာ ပေးထားသည် သို့မဟုတ် လုံးဝ ပေးမထားပါ။

သင့်ကုဒ်ကို စာရိုက်နေစဉ် သို့မဟုတ် ဖိုင်ကို သိမ်းဆည်းသည့်အခါ auto-format ပြုလုပ်ပေးနိုင်ရန် code formatter ကို **IDE integration** (https://missing-semester-my.github.io/2026/development-environment/#code-intelligence-and-language-servers) တွင် ထည့်သွင်း သတ်မှတ်ထားနိုင်ပါသည်။ ဖိုင်အမျိုးအစား အသီးသီးအတွက် indent size ကဲ့သို့သော ပရောဂျက်အဆင့် setting များကို မိမိ၏ IDE သို့ အကြောင်းကြားပေးသည့် **EditorConfig** (https://editorconfig.org/) ဖိုင်တစ်ခုကိုလည်း မိမိပရောဂျက်တွင် ထည့်သွင်းထားနိုင်ပါသည်။

Linting

Linter များသည် သင့်ကုဒ်ကို run စရာ မလိုဘဲ စစ်ဆေးသည့် static analysis စနစ်ကို အသုံးပြု၍ ကုဒ်အတွင်းရှိ antipattern များနှင့် ဖြစ်လာနိုင်ဖွယ်ရှိသော ပြဿနာများကို ရှာဖွေပေးပါသည်။ ဤ tool များသည် ရိုးရိုး syntax လောက်သာ မဟုတ်ဘဲ autoformatter များထက် ပိုမို နက်နဲစွာ စစ်ဆေးပေးပါသည်။ စစ်ဆေးမှု၏ နက်နဲမှုမှာမူ tool အလိုက် ကွဲပြားမှု ရှိပါသည်။

Linter များတွင် စည်းမျဉ်းများ (*rules*) စာရင်း ပါဝင်ပြီး ပရောဂျက်အဆင့်အလိုက် စိတ်ကြိုက် ပြင်ဆင်နိုင်သော preset များ ပါဝင်ပါသည်။ အချို့သော linter rule များသည် false positive (မှားယွင်း အချက်ပေးခြင်း) များကို ဖြစ်ပေါ်စေနိုင်သဖြင့် ယင်းတို့ကို ဖိုင်တစ်ခုချင်းစီအလိုက် သို့မဟုတ် လိုင်းတစ်ခုချင်းစီအလိုက် disable လုပ်ထားနိုင်ပါသည်။

ကောင်းမွန်သော linter များတွင် linter rule တစ်ခုချင်းစီ၏ သဘောတရား — ထို rule က မည်သည့်အရာကို ရှာဖွေနေသနည်း၊ အဘယ်ကြောင့် မကောင်းသနည်း၊ ထို ကုဒ် pattern အတွက် ပိုမို ကောင်းမွန်သည့် အခြားရွေးချယ်စရာမှာ မည်သည့်အရာ ဖြစ်သနည်း စသည်တို့ကို ရှင်းပြထားသည့် Documentation သို့မဟုတ် အကူအညီများ built-in ပါရှိပါသည်။ ဥပမာအားဖြင့် Python ကုဒ်တွင် မလိုအပ်ဘဲ ထပ်နေသော `if` statement များကို ဖမ်းထုတ်ပေးသည့် **Ruff** (https://docs.astral.sh/ruff/) ထဲရှိ **SIM102** (https://docs.astral.sh/ruff/rules/collapsible-if/) rule ၏ documentation ကို ကြည့်ရှုနိုင်ပါသည်။

အချို့သော linter များသည် ပြဿနာများကို အချက်ပြရုံသာမက သတ်မှတ်ထားသော ပြဿနာ အချို့ကိုပါ သင့်အတွက် အလိုအလျောက် ပြင်ဆင်ပေးနိုင်ပါသည်။

Language အလိုက် သီးခြား linter များအပြင် အသုံးဝင်နိုင်သည့် အခြား tool တစ်ခုမှာ **semgrep** (https://github.com/semgrep/semgrep) ဖြစ်ပါသည်။ ၎င်းသည် (grep ကဲ့သို့ character အဆင့် မဟုတ်ဘဲ) AST (Abstract Syntax Tree) အဆင့်တွင် အလုပ်လုပ်သည့် “semantic grep” tool တစ်ခုဖြစ်ပြီး language အများအပြားကို ပံ့ပိုးပေးပါသည်။ သင့် ပရောဂျက်များအတွက် custom linter rule များကို လွယ်ကူစွာ ရေးသားရန် semgrep ကို အသုံးပြုနိုင်ပါသည်။ ဥပမာအားဖြင့် Python တွင် အန္တရာယ်ရှိသော `subprocess.Popen(..., shell=True)` အသုံးပြုမှုကို တားဆီးလိုပါက ထို ကုဒ် pattern ကို အောက်ပါအတိုင်း ရှာဖွေနိုင်ပါသည် -

```
semgrep -l python -e "subprocess.Popen(..., shell=True, ...)"
```

Testing

Software testing သည် သင့်ကုဒ်၏ မှန်ကန်မှုအပေါ် ယုံကြည်မှု တိုးပွားစေရန် ဆောင်ရွက်သည့် စံနှုန်းမီ နည်းလမ်းတစ်ခု ဖြစ်ပါသည်။ သင်သည် ကုဒ် ရေးသားပြီးနောက် ထိုရေးသားထားသော ကုဒ်ကို စမ်းသပ် စစ်ဆေးသည့် ကုဒ်ကို ရေးသားရမည်ဖြစ်ပြီး၊ မျှော်မှန်းထားသည့်အတိုင်း အလုပ်မလုပ်ပါက error အချက်ပေးအောင် ပြုလုပ်ရမည် ဖြစ်ပါသည်။

ကုဒ် အစိတ်အပိုင်းများအတွက် test များကို အသေးစိတ် အဆင့်အမျိုးမျိုးဖြင့် ရေးသားနိုင်ပါသည် - သီးခြား function တစ်ခုချင်းစီအတွက် *unit tests*၊ module သို့မဟုတ် service များအကြား ဓာတ်ပြု အလုပ်လုပ်ပုံ အတွက် *integration tests*၊ နှင့် အစမှ အဆုံးအထိ လုပ်ငန်းစဉ်များ (end-to-end scenarios) အတွက် *functional tests* တို့ ဖြစ်ကြပါသည်။ Implementation ကုဒ်များ မရေးမီ test များကို ဦးစွာ ရေးသားသည့် *test-driven development* ကိုလည်း ပြုလုပ်နိုင်ပါသည်။ သင့်ကုဒ်တွင် bug များကို တွေ့ရှိပါက နောက်ပိုင်းတွင် ထိုလုပ်ဆောင်ချက်များ မပျက်စီးစေရန် စစ်ဆေးပေးသည့် *regression tests* များကို ရေးသားနိုင်ပါသည်။ Haskell ၏ **QuickCheck** (https://hackage.haskell.org/package/QuickCheck) တွင် စတင်ခဲ့ပြီး Python အတွက် **Hypothesis** (https://hypothesis.readthedocs.io/) ကဲ့သို့သော library အမြောက်အမြားတွင် အသုံးပြုထားသည့် *property-based tests* များကိုလည်း ရေးသားနိုင်ပါသည်။ မည်သည့် testing နည်းလမ်းက သင့်တော်သနည်း ဆိုသည်မှာ သင့် ပရောဂျက်ပေါ်တွင် မူတည်ပြီး နည်းလမ်း အချို့ကို ပေါင်းစပ် အသုံးပြုရလေ့ ရှိပါသည်။

သင့် ပရိုဂရမ်တွင် database သို့မဟုတ် web API ကဲ့သို့သော ပြင်ပမှ ပံ့ပိုးမှုများ (external dependencies) ပါဝင်နေပါက test ပြုလုပ်ချိန်တွင် ပြင်ပ ပံ့ပိုးမှုများနှင့် တိုက်ရိုက် ချိတ်ဆက် အလုပ်လုပ်စေမည့်အစား ထို dependencies များကို *mock* ပြုလုပ် (တုပထား) ခြင်းက အထောက်အကူ ဖြစ်စေနိုင်ပါသည်။

Code coverage

Code coverage သည် သင့် test များ မည်မျှ ကောင်းမွန်ကြောင်း တိုင်းတာနိုင်သည့် တိုင်းတာမှု စံနှုန်း (metric) တစ်ခု ဖြစ်ပါသည်။ Code coverage သည် သင့် test များကို run သည့်အခါ မည်သည့် ကုဒ်လိုင်းများ အလုပ်လုပ်သွားသည်ကို ကြည့်ရှုစစ်ဆေးသဖြင့် ကုဒ် လမ်းကြောင်း အားလုံးကို လွှမ်းခြုံနိုင်စေရန် သေချာစေပါသည်။ Code coverage tool များသည် test များ ရေးသားရာတွင် လမ်းညွှန်ပေးနိုင်ရန် လိုင်းတစ်ခုချင်းစီ၏ coverage များကို ပြသပေးနိုင်ပါသည်။ **Codecov** (https://app.codecov.io) ကဲ့သို့သော service များသည် ပရောဂျက်တစ်ခု၏ သမိုင်းကြောင်းတစ်လျှောက် code coverage ကို စောင့်ကြည့်ရန်နှင့် ကြည့်ရှုရန် web interface များကို ထောက်ပံ့ပေးပါသည်။

မည်သည့် တိုင်းတာမှု စံနှုန်းမဆို ကောင်းမွန်ပြည့်စုံခြင်း မရှိသကဲ့သို့ code coverage သည်လည်း အပြည့်အ
ဝ မမှန်ကန်နိုင်ပါ သို့ဖြစ်၍ coverage အပေါ်တွင်သာ အလွန်အမင်း အာရုံမစိုက်ဘဲ အရည်အသွေး
မြင့်မားသော test များကို ရေးသားရန်သာ အဓိက ထားသင့်ပါသည်။

Pre-commit hooks

pre-commit (https://pre-commit.com/) framework ကြောင့် ပိုမို လွယ်ကူလာသည့် Git pre-commit **hooks** (https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks) များသည် Git commit မပြုလုပ်မီ အချို့နှိုင်းတိုင်းတွင် အသုံးပြုသူ သတ်မှတ်ထားသော ကုဒ်များကို အလိုအလျောက် run ပေးပါသည်။ commit ပြုလုပ်လိုက်သော ကုဒ်များသည် ပရောဂျက်၏ ကုဒ်စတိုင်လ်နှင့် ကိုက်ညီမှုရှိပြီး သတ်မှတ်ထားသော ပြဿနာများ မရှိစေရန် သေချာစေရေးအတွက် commit တိုင်း မတိုင်မီ formatter များ၊ linter များနှင့် အချို့သော test များကို အလိုအလျောက် run ရန် ပရောဂျက်များတွင် pre-commit hook များကို ယေဘုယျအားဖြင့် အသုံးပြုကြပါသည်။

Continuous integration

GitHub Actions (https://github.com/features/actions) ကဲ့သို့သော Continuous integration (CI) service များသည် သင် ကုဒ် push လုပ်သည့် အကြိမ်တိုင်းတွင် (သို့မဟုတ် pull request တိုင်းတွင် သို့မဟုတ် အချိန်ဇယားအတိုင်း) script များကို run ပေးနိုင်ပါသည်။ Developer များသည် formatter များ၊ linter များနှင့် test များ အပါအဝင် ကုဒ် အရည်အသွေး ထိန်းသိမ်းသည့် tool များကို run ရန်အတွက် CI service များကို ယေဘုယျအားဖြင့် အသုံးပြုကြပါသည်။ Compiled language များအတွက် ကုဒ် compile ဖြစ်မဖြစ် စစ်ဆေးနိုင်ပြီး statically typed language များအတွက် type စစ်ဆေးမှု မှန်မမှန် စစ်ဆေးနိုင်ပါသည်။ Commit အသစ်များ push လုပ်သည့် အကြိမ်တိုင်း CI ကို run ပေးခြင်းဖြင့် ပင်မ ကုဒ်ထဲသို့ အမှားများ ပါဝင်သွားခြင်းကို တားဆီးပေးနိုင်သည်၊ pull request များတွင် run ပေးခြင်းဖြင့် ကူညီ ပံ့ပိုးသူများ၏ ပေးပို့ချက်များမှ ပြဿနာများကို ဖမ်းထုတ်ပေးနိုင်သည်၊ သတ်မှတ် အချိန်ဇယားအတိုင်း run ပေးခြင်းဖြင့် ပြင်ပ ပံ့ပိုးမှုများမှ ပြဿနာများကို ဖမ်းထုတ်ပေးနိုင်ပါသည် (ဥပမာ - Developer တစ်ဦးက သဟဇာတမဖြစ်သော ပြောင်းလဲမှုတစ်ခုကို **semver-compatible** (https://missing-semester-my.github.io/2026/shipping-code/#releases–versioning) အဖြစ် မှားယွင်း ထုတ်လုပ်လိုက်ခြင်း ကဲ့သို့သော)။

CI script များသည် Developer များ၏ စက်များနှင့် သီးခြားစီ run သောကြောင့် အချိန်အကြာကြီး run ရမည့် အလုပ်များကို ထိုနေရာတွင် လွယ်ကူစွာ run နိုင်ပါသည်။ ဥပမာအားဖြင့် ဆော့ဖ်ဝဲသည် operating system များနှင့် programming language version မျိုးစုံတွင် မှန်ကန်စွာ အလုပ်လုပ်ကြောင်း သေချာစေရန် test အမြောက်အများ၏ *matrix* အလိုက် run ရာတွင် ဤအချက်ကို အသုံးချနိုင်ပါသည်။

ယေဘုယျအားဖြင့် CI တွင် run သော script သည် သင့်ကုဒ်ကို တိုက်ရိုက် ပြင်ဆင်မည် မဟုတ်ပါ - ၎င်းသည် tool များကို "fix" mode အစား "check-only" mode ဖြင့်သာ run မည်ဖြစ်ရာ ဥပမာအားဖြင့် ကုဒ်သည် သတ်မှတ်ပုံစံနှင့် မကိုက်ညီပါက auto-formatter က error အချက်ပေးမည် ဖြစ်ပါသည်။

Repository အများအပြားသည် ၎င်းတို့၏ README တွင် CI အခြေအနေနှင့် code coverage ကဲ့သို့သော အခြား အချက်အလက်များကို ပြသသည့် **status badge** (https://docs.github.com/en/actions/how-tos/monitor-workflows/add-a-status-badge) များကို ထည့်သွင်းထားလေ့ ရှိကြပါသည်။ ဥပမာအားဖြင့် အောက်တွင် ဖော်ပြထားသည်မှာ Missing Semester ၏ လက်ရှိ build status ဖြစ်ပါသည်။

Build Status Links Status

> **proof-html** (https://github.com/anishathalye/proof-html) GitHub Action ကို အသုံးပြုထားသည့် ကျွန်ုပ်တို့၏ **links checker** (https://github.com/missing-semester/missing-semester/blob/master/.github/workflows/links.yml) သည် ပြင်ပ ဝဘ်ဆိုက်များ၏ ပြဿနာများကြောင့် မကြာခဏ အလုပ်မလုပ်ဘဲ ဖြစ်တတ်ပါသည်။ သို့သော်လည်း ၎င်းသည် ပျက်စီးနေသော link အမြောက်အများကို ဖမ်းထုတ်၍ ပြင်ဆင်နိုင်ရန် ကူညီပေးခဲ့ပါသည် (အချို့မှာ စာလုံးပေါင်းမှားယွင်းမှုကြောင့် ဖြစ်ပြီး အများစုမှာ ဝဘ်ဆိုက်များက redirect မလုပ်ဘဲ အကြောင်းအရာများကို ရွှေ့ပြောင်းလိုက်ခြင်း သို့မဟုတ် ဝဘ်ဆိုက်များ ပျောက်ကွယ်သွားခြင်း ကြောင့် ဖြစ်သည်)။

CI service များ၊ formatter များ၊ linter များနှင့် testing library များနှင့် ပတ်သက်သော အသေးစိတ် အချက်အလက်များကို လေ့လာရန် ကောင်းမွန်သော နည်းလမ်းတစ်ခုမှာ နမူနာများကို ကြည့်ရှုခြင်း ဖြစ်ပါသည်။ GitHub တွင် အရည်အသွေးမြင့် open-source ပရောဂျက်များကို ရှာဖွေပါ — သင့် ပရောဂျက်နှင့် programming language၊ နယ်ပယ်၊ အရွယ်အစားနှင့် နယ်ပယ် အတိုင်းအတာ စသည်ဖြင့် ပိုမို တူညီလေလေ ပိုမို ကောင်းမွန်လေလေ ဖြစ်သည် — ပြီးလျှင် ၎င်းတို့၏ `pyproject.toml` ၊ `.github/workflows/` ၊ `DEVELOPMENT.md` နှင့် အခြား သက်ဆိုင်ရာ ဖိုင်များကို လေ့လာပါ။

Continuous deployment

Continuous deployment သည် ပြောင်းလဲမှုများကို လက်တွေ့ *deploy* လုပ်ရန်အတွက် CI အခြေခံ အဆောက်အအုံကို အသုံးပြုပါသည်။ ဥပမာအားဖြင့် Missing Semester repository သည် GitHub pages သို့ continuous deployment ကို အသုံးပြုထားရာ ကျွန်ုပ်တို့မှ မွမ်းမံထားသော သင်ခန်းစာ မှတ်စုများကို `git push` လုပ်လိုက်သည်နှင့် ဝဘ်ဆိုက်ကို အလိုအလျောက် build ပြီး deploy လုပ်ပေးပါသည်။ Application များအတွက် binary များ သို့မဟုတ် service များအတွက် Docker image များ ကဲ့သို့သော အခြားသော **artifact** (https://missing-semester-my.github.io/2026/shipping-code/) အမျိုးအစားများကိုလည်း CI တွင် build နိုင်ပါသည်။

Command runners

just (https://github.com/casey/just) ကဲ့သို့သော Command runner များသည် ပရောဂျက်တစ်ခုတွင် command များကို run သည့် အလုပ်ကို ပိုမို လွယ်ကူစေပါသည်။ သင့်ပရောဂျက်အတွက် ကုဒ် အရည်အသွေး ထိန်းသိမ်းမှု အခြေခံအဆောက်အအုံကို တည်ဆောက်သည့်အခါ သင့် Developer များကို `uv run ruff check --fix` ကဲ့သို့သော command များကို အလွတ်ကျက်ခိုင်းရန် မလိုတော့ပါ။ Command runner တစ်ခုဖြင့် ၎င်းကို `just lint` ဟု အပြောင်းအလဲ ပြုလုပ်နိုင်ပြီး Developer တစ်ဦး မိမိပရောဂျက်အတွက် run လိုသည့် tool မျိုးစုံအတွက် `just format` ၊ `just typecheck` စသည်ဖြင့် အလားတူ ခေါ်ယူမှုများကို ထားရှိနိုင်ပါသည်။

Language အလိုက် သီးခြား ပရောဂျက် သို့မဟုတ် package manager အချို့တွင် ယင်းလုပ်ဆောင်ချက်အတွက် built-in ပံ့ပိုးမှု ပါဝင်သောကြောင့် `just` ကဲ့သို့သော ရိုးရိုး tool များကို အသုံးပြုရန် မလိုတော့ပါ။ ဥပမာအားဖြင့် **npm** (https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager) (Node.js) အတွက် `package.json` ၏ `scripts` အပိုင်းနှင့် **Hatch** (https://hatch.pypa.io/) (Python) အတွက် `pyproject.toml` ၏ `tool.hatch.envs.*.scripts` အပိုင်းတို့သည် ဤလုပ်ဆောင်ချက်ကို ပံ့ပိုးပေးကြပါသည်။

Regular expressions

အတိုကောက်အားဖြင့် “regex” ဟု ခေါ်လေ့ရှိသော *Regular expressions* သည် string အစုအဝေးများကို ကိုယ်စားပြုရန် အသုံးပြုသည့် ဘာသာစကား တစ်ခု ဖြစ်ပါသည်။ Regex pattern များကို command-line tool များနှင့် IDE များကဲ့သို့သော အခြေအနေ အမျိုးမျိုးတွင် pattern matching ပြုလုပ်ရန် အသုံးပြုကြပါသည်။ ဥပမာအားဖြင့် **ag** (https://github.com/ggreer/the_silver_searcher) သည် codebase တစ်ခုလုံးတွင် ရှာဖွေမှု ပြုလုပ်ရန် regex pattern များကို ပံ့ပိုးပေးပါသည် (ဥပမာ - `ag "import .* as .*"` သည် Python ရှိ အမည်ပြောင်းလဲထားသော import အားလုံးကို ရှာဖွေပေးမည်ဖြစ်သည်)။ ထို့ပြင် **go test** (https://pkg.go.dev/cmd/go#hdr-Test_packages) သည် test အစီတ်အပိုင်းများကို ရွေးချယ်ရန်အတွက် `-run [regexp]` option ကို ပံ့ပိုးပေးပါသည်။ ထို့ပြင် programming language များတွင် regular expression matching အတွက် built-in ပံ့ပိုးမှု သို့မဟုတ် ပြင်ပ library များ ပါရှိသောကြောင့် pattern matching၊ validation နှင့် parsing ကဲ့သို့သော လုပ်ဆောင်ချက်များအတွက် regex များကို အသုံးပြုနိုင်ပါသည်။

နားလည်သဘောပေါက်လွယ်စေရန် အောက်တွင် regex pattern နမူနာအချို့ကို ဖော်ပြထားပါသည်။ ဤသင်ခန်းစာတွင် ကျွန်ုပ်တို့သည် **Python regex syntax** (https://docs.python.org/3/library/re.html) ကို အသုံးပြုထားပါသည်။ Regex တွင် အမျိုးအစား (flavor) များစွာ ရှိပြီး ၎င်းတို့အကြားတွင် အထူးသဖြင့် ပိုမို ရှုပ်ထွေးသော လုပ်ဆောင်ချက်များ၌ ကွဲပြားမှု အနည်းငယ် ရှိကြပါသည်။ Regular expression များကို ရေးသားရန်နှင့် debug ပြုလုပ်ရန် **regex101** (https://regex101.com/) ကဲ့သို့သော အွန်လိုင်း regex tester များကို အသုံးပြုနိုင်ပါသည်။

- `abc` — တိုက်ရိုက် စာသား “abc” နှင့် ကိုက်ညီပါသည်။
- `missing|semester` — “missing” သို့မဟုတ် “semester” စာသားနှင့် ကိုက်ညီပါသည်။
- `\d{4}-\d{2}-\d{2}` — “2026-01-14” ကဲ့သို့သော YYYY-MM-DD ပုံစံရှိ ရက်စွဲများနှင့် ကိုက်ညီပါသည်။ စာသားတွင် ကိန်းဂဏန်း ၄ လုံး၊ dash တစ်ခု၊ ကိန်းဂဏန်း ၂ လုံး၊ dash တစ်ခု နှင့် ကိန်းဂဏန်း ၂ လုံး ပါဝင်ကြောင်း စစ်ဆေးရုံမျှမက ဤ regex သည် ရက်စွဲ အမှန်အကန် ဟုတ်မဟုတ် စိစစ်ပေးမည် မဟုတ်ပါ၊ သို့ဖြစ်၍ “2026-01-99” သည်လည်း ဤ regex pattern နှင့် ကိုက်ညီနေမည် ဖြစ်ပါသည်။
- `.+@.+` — အီးမေးလ် လိပ်စာများ၊ စာသားအချို့ ပါဝင်ပြီး နောက်တွင် “@” ပါကာ နောက်တွင် စာသားအချို့ ထပ်မံ ပါဝင်သော စာသားများနှင့် ကိုက်ညီပါသည်။ ၎င်းသည် အခြေခံကျသော စစ်ဆေးမှုကိုသာ ပြုလုပ်ပေးပြီး “nonsense@@@email” ကဲ့သို့သော စာသားများနှင့် ကိုက်ညီနေမည် ဖြစ်ပါသည်။ False positive သို့မဟုတ် false negative လုံးဝ မရှိဘဲ အီးမေးလ် လိပ်စာများကို စစ်ဆေးပေးသည့် regex **ရှိသော်လည်း** (https://pdw.ex-parrot.com/Mail-RFC822-Address.html) လက်တွေ့တွင် အသုံးပြုရန် မလွယ်ကူပါ။

Regex syntax

Regex syntax ဆိုင်ရာ အပြည့်အစုံ လမ်းညွှန်ကို ဤ **documentation** (https://docs.python.org/3/library/re.html#regular-expression-syntax) တွင် (သို့မဟုတ် အွန်လိုင်းတွင် ရရှိနိုင်သော အခြား အရင်းအမြစ်များတွင်) ရှာဖွေနိုင်ပါသည်။ အဓိက အခြေခံ အစိတ်အပိုင်း အချို့မှာ အောက်ပါအတိုင်း ဖြစ်ကြ ပါသည် -

- `abc` သည် အက္ခရာများတွင် အထူး အဓိပ္ပာယ် မရှိသည့်အခါ တိုက်ရိုက် စာသားနှင့် ကိုက်ညီပါသည် (ဤ နမူနာတွင် “abc”)
- `.` သည် မည်သည့် single character မဆို ၎င်းနှင့် ကိုက်ညီပါသည်
- `[abc]` သည် bracket အတွင်း ပါရှိသော single character နှင့် ကိုက်ညီပါသည် (ဤနမူနာတွင် “a”, “b”, သို့မဟုတ် “c”)
- `[^abc]` သည် bracket အတွင်း ပါရှိသော character များမှလွဲ၍ ကျန် single character များနှင့် ကိုက် ညီပါသည် (ဥပမာ - “d”)
- `[a-f]` သည် bracket အတွင်း ဖော်ပြထားသော ပမာဏ အကွာအဝေးအတွင်း ပါရှိသော single character နှင့် ကိုက်ညီပါသည် (ဥပမာ - “c”, သို့သော် “q” မဟုတ်ပါ)
- `a|b` သည် pattern နှစ်ခုအနက် တစ်ခုခုနှင့် ကိုက်ညီပါသည် (ဥပမာ - “a” သို့မဟုတ် “b”)
- `\d` သည် မည်သည့် ဂဏန်း (digit) character မဆို ၎င်းနှင့် ကိုက်ညီပါသည် (ဥပမာ - “3”)
- `\w` သည် မည်သည့် word character မဆို ၎င်းနှင့် ကိုက်ညီပါသည် (ဥပမာ - “x”)
- `\b` သည် မည်သည့် စကားလုံး နယ်နိမိတ် (*boundary*) မဆို ၎င်းနှင့် ကိုက်ညီပါသည် (ဥပမာ - “missing semester” ဟူသော စာသားတွင် “m” ၏ အရှေ့၊ “g” ၏ အနောက်၊ “s” ၏ အရှေ့၊ နှင့် “r” ၏ အနောက်တို့နှင့် ကိုက်ညီပါသည်)
- `(...)` သည် pattern တစ်ခု၏ အုပ်စု (group) နှင့် ကိုက်ညီပါသည်
- `...?` သည် pattern တစ်ခု၏ သုည သို့မဟုတ် တစ်ခုနှင့် ကိုက်ညီပါသည်၊ ဥပမာ - “word” သို့မဟုတ် “words” နှင့် ကိုက်ညီစေရန် `words?`
- `...*` သည် pattern တစ်ခု၏ မည်သည့် အရေအတွက်မဆို ကိုက်ညီပါသည်၊ ဥပမာ - မည်သည့် character ၏ မည်သည့် အရေအတွက်မဆို ကိုက်ညီစေရန် `.*`
- `...+` သည် pattern တစ်ခု၏ တစ်ခု သို့မဟုတ် တစ်ခုထက်ပိုသော အရေအတွက်နှင့် ကိုက်ညီပါသည်၊ ဥပမာ - သုည မဟုတ်သော ဂဏန်း အရေအတွက်မဆို ကိုက်ညီစေရန် `\d+`
- `...{N}` သည် pattern တစ်ခု၏ တိကျသော အရေအတွက် N ခုနှင့် ကိုက်ညီပါသည်၊ ဥပမာ - ဂဏန်း ၄ လုံးအတွက် `\d{4}`

- `\.` သည် တိုက်ရိုက် စာသား “.” နှင့် ကိုက်ညီပါသည်
- `\\` သည် တိုက်ရိုက် စာသား “\” နှင့် ကိုက်ညီပါသည်
- `^` သည် လိုင်း၏ အစနှင့် ကိုက်ညီပါသည်
- `$` သည် လိုင်း၏ အဆုံးနှင့် ကိုက်ညီပါသည်

Capture groups နှင့် references

Regex group `(...)` ကို အသုံးပြုပါက ရှာဖွေရန် သို့မဟုတ် အစားထိုးရန် အတွက် match ၏ အစိတ်အပိုင်းများကို ညွှန်းဆိုနိုင်ပါသည်။ ဥပမာအားဖြင့် YYYY-MM-DD ပုံစံ ရက်စွဲတစ်ခုမှ လ (month) ကို သာ ထုတ်ယူရန်အတွက် အောက်ပါ Python ကုဒ်ကို အသုံးပြုနိုင်ပါသည် -

```
>>> import re
>>> re.match(r"\d{4}-(\d{2})-\d{2}", "2026-01-14").group(1)
'01'
```

သင့် text editor တွင် replace pattern များထဲ၌ reference capture group များကို အသုံးပြုနိုင်ပါသည်။ Syntax သည် IDE အလိုက် ကွဲပြားနိုင်ပါသည်။ ဥပမာအားဖြင့် VS Code တွင် group များကို ညွှန်းဆိုရန် `$1`, `$2` စသည်ဖြင့် သုံးနိုင်ပြီး Vim တွင် `\1`, `\2` စသည်ဖြင့် သုံးနိုင်ပါသည်။

Limitations

Regular language (https://en.wikipedia.org/wiki/Regular_language) များသည် စွမ်းဆောင်ရည် မြင့်မားသော်လည်း အကန့်အသတ် ရှိကြပါသည်; ရိုးရိုး regex အဖြစ် ဖော်ပြ၍ မရနိုင်သော စာသား အမျိုးအစားများ ရှိပါသည် (ဥပမာ - “a” အရေအတွက် အတိုင်း အနောက်တွင် “b” အရေအတွက် ပါဝင်သော {a^n b^n | n ≥ 0} စာသား အစုအဝေးကို ကိုက်ညီစေမည့် regular expression ရေးသားရန် **မဖြစ်နိုင်ပါ** (https://en.wikipedia.org/wiki/Pumping_lemma_for_regular_languages); လက်တွေ့တွင် HTML ကဲ့သို့သော ဘာသာစကားများသည် regular language များ မဟုတ်ကြပါ)။ လက်တွေ့တွင် ခေတ်မီ regex engine များသည် regular language များထက် ကျော်လွန်၍ lookahead နှင့် backreference ကဲ့သို့သော လုပ်ဆောင်ချက်များကို ပံ့ပိုးပေးထားသဖြင့် လက်တွေ့တွင် အလွန် အသုံးဝင်သော်လည်း ၎င်းတို့၏ ဖော်ပြနိုင်စွမ်းတွင် အကန့်အသတ် ရှိနေဆဲဖြစ်ကြောင်း သိရှိထားရန် အရေးကြီးပါသည်။ ပိုမို ရှုပ်ထွေးသော ဘာသာစကားများအတွက် ပိုမို စွမ်းဆောင်နိုင်သည့် parser အမျိုးအစားများကို အသုံးပြုရန် လိုအပ်နိုင်ပါသည် (ဥပမာတစ်ခုအနေဖြင့် **pyparsing** (https://github.com/pyparsing/pyparsing) ဟူသော **PEG** (https://en.wikipedia.org/wiki/Parsing_expression_grammar) parser ကို ကြည့်ပါ)။

Learning regex

ဘာသာစကား တစ်ခုလုံးကို အလွတ်ကျက်မှတ်နေမည့်အစား အခြေခံများကို လေ့လာပြီး (ဤသင်ခန်းစာတွင် ကျွန်ုပ်တို့ လွှမ်းခြုံထားသကဲ့သို့) လိုအပ်သည့်အခါမှသာ regex reference များကို ကြည့်ရှုရန် အကြံပြုလိုပါသည်။

Conversational AI tool များသည် regex pattern များ ထုတ်လုပ်ရာတွင် ကူညီပေးရန် ထိရောက်မှု ရှိနိုင်ပါသည်။ ဥပမာအားဖြင့် မိမိ နှစ်သက်ရာ LLM ကို အောက်ပါ query ဖြင့် prompt ပေးကြည့်ပါ -

```
Write a Python-style regex pattern that matches the requested path from log lin
es from Nginx. Here is an example log line:

169.254.1.1 - - [09/Jan/2026:21:28:51 +0000] "GET /feed.xml HTTP/2.0" 200 2995
"-" "python-requests/2.32.3"
```

လေ့ကျင့်ခန်းများ

1. သင် အလုပ်လုပ်နေသော ပရောဂျက်တစ်ခုအတွက် formatter၊ linter နှင့် pre-commit hook များကို သတ်မှတ်ပြင်ဆင်ပါ။ အကယ်၍ အများများစွာ ရှိနေပါက autoformatting က format အများများကို ရှင်းလင်းပေးပါလိမ့်မည်။ Linter အများများအတွက်မူ linter အများအားလုံးကို ပြင်ဆင်ရန် **AI agent** (https://missing-semester-my.github.io/2026/agentic-coding/) ကို အသုံးပြုကြည့်ပါ။ ပြဿနာအားလုံးကို ထပ်ခါတလဲလဲ ပြင်ဆင်နိုင်ရန် AI agent အနေဖြင့် linter ကို run နိုင်ပြီး ရလဒ်များကို လေ့လာနိုင်စေရန် သေချာပါစေ။ AI က သင့်ကုဒ်ကို ပျက်စီးမသွားစေရန် ရလဒ်များကို သေချာစွာ စစ်ဆေးပါ!

2. သင်သိသော ဘာသာစကားတစ်ခုအတွက် testing library တစ်ခုကို လေ့လာပြီး သင် အလုပ်လုပ်နေသော ပရောဂျက်တစ်ခုအတွက် unit test တစ်ခု ရေးသားပါ။ Code coverage tool တစ်ခုကို run ပါ၊ HTML-formatted coverage report တစ်ခု ထုတ်ယူပြီး ရလဒ်များကို လေ့လာပါ။ လွှမ်းခြုံထားသော လိုင်းများကို ရှာဖွေနိုင်ပါသလား? သင့် code coverage သည် အလွန် နည်းပါးနိုင်ပါသည်။ ၎င်းကို မြှင့်တင်ရန် test အချို့ကို ကိုယ်တိုင် ရေးသားကြည့်ပါ။ Coverage ကို မြှင့်တင်ရန် **AI agent** (https://missing-semester-my.github.io/2026/agentic-coding/) ကို အသုံးပြုကြည့်ပါ; coding agent အနေဖြင့် စိစစ်ရမည့်နေရာကို သိရှိစေရန် coverage ဖြင့် test များကို run နိုင်ပြီး လိုင်းတစ်ခုချင်းစီ၏ coverage report ကို ထုတ်ယူနိုင်ကြောင်း သေချာပါစေ။ AI က ထုတ်လုပ်ပေးသော test များသည် အမှန်တကယ် ကောင်းမွန်ပါသလား?

3. သင် အလုပ်လုပ်နေသော ပရောဂျက်တစ်ခုအတွက် push လုပ်သည့် အကြိမ်တိုင်းတွင် run ရန် continuous integration ကို သတ်မှတ်ပါ။ CI ကို formatting၊ linting နှင့် test များကို run ခိုင်းပါ။ သင့်ကုဒ်ကို တမင် ပျက်စီးအောင် ပြုလုပ်ကြည့်ပါ (ဥပမာ - linter စည်းမျဉ်းကို ချိုးဖောက်ကြည့်ပါ)၊ CI က ၎င်းကို ဖမ်းထုတ်နိုင်ကြောင်း သေချာပါစေ။

4. **regex pattern** တစ်ခု ရေးသားကြည့်ပြီး သင့်ကုဒ်အတွင်းရှိ `subprocess.Popen(..., shell=True)` တွေ့ရှိချက်များကို ရှာဖွေရန် `grep` **command-line tool** (https://missing-semester-my.github.io/2026/course-shell/) ကို အသုံးပြုပါ။ ယခုအခါ regex pattern ကို “ပျက်စီးအောင်” ပြုလုပ်ကြည့်ပါ။ သင့် grep ခေါ်ယူမှုကို အဟန့်အတားဖြစ်စေသည့် အန္တရာယ်ရှိသော ကုဒ်ကို **semgrep** က အောင်မြင်စွာ ကိုက်ညီအောင် ရှာဖွေပေးနိုင်သေးသလား?

5. **ဤသင်ခန်းစာ မှတ်စုများ** (https://raw.githubusercontent.com/missing-semester/missing-semester/refs/heads/master/_2026/code-quality.md) ထဲရှိ `-` **Markdown bullet marker** (https://spec.commonmark.org/0.31.2/#bullet-list-marker) များကို `*` bullet marker များဖြင့် အစားထိုးခြင်းဖြင့် သင့် IDE သို့မဟုတ် text editor တွင် regex search-and-replace ကို လေ့ကျင့်ပါ။ ဖိုင်အတွင်းရှိ “-” အက္ခရာ အားလုံးကို အစားထိုးလိုက်ပါက မှားယွင်းသွားမည်ဖြစ်ကြောင်း သတိပြုပါ၊ အကြောင်းမှာ ထို အက္ခရာသည် bullet marker မဟုတ်ဘဲ အခြားနေရာများတွင်လည်း သုံးထားသောကြောင့် ဖြစ်ပါသည်။

6. `{"name": "Alyssa P. Hacker", "college": "MIT"}` ပုံစံရှိသော JSON structure ထဲမှ အမည် (ဥပမာ - ဤနမူနာတွင် `Alyssa P. Hacker`) ကို ဖမ်းယူရန် regex တစ်ခု ရေးသားပါ။ လမ်းညွှန်: သင်၏ ပထမဆုံး ကြိုးပမ်းမှုတွင် `Alyssa P. Hacker", "college": "MIT` ကို ထုတ်ယူသည့် regex မျိုး ရေးမိသွားနိုင်သည်; ၎င်းကို ပြင်ဆင်ပုံအား လေ့လာရန် [Python regex docs](https://docs.python.org/3/library/re.html) ရှိ greedy quantifier များအကြောင်း ဖတ်ရှုပါ။

1. အမည်တွင် `"` အက္ခရာ ပါဝင်နေသည့် အခြေအနေမျိုးတွင်ပင် regex pattern ကို အလုပ်လုပ်အောင် ပြုလုပ်ပါ (JSON တွင် double quote များကို `\"` ဖြင့် escape ပြုလုပ်နိုင်ပါသည်)။
1. လက်တွေ့တွင် ပိုမို ရှုပ်ထွေးသော parsing ပြဿနာများအတွက် regular expression များကို အသုံးပြုရန် ကျွန်ုပ်တို့ **အကြိမ်ပြုပါ**။ ဤအလုပ်အတွက် သင့် programming language ၏ JSON parser ကို မည်သို့ အသုံးပြုရမည်ကို ရှာဖွေပါ။ အထက်ပါ ဖော်ပြပါ JSON structure ကို stdin မှ input အဖြစ် ရယူပြီး stdout သို့ အမည်ကို ထုတ်ပေးသည့် command-line program တစ်ခု ရေးသားပါ။ ဤသို့ ပြုလုပ်ရန် ကုဒ်အနည်းငယ်မျှသာ လိုအပ်မည် ဖြစ်ပါသည်။ Python တွင် `import json` အပြင် ကုဒ်တစ်လိုင်းတည်းဖြင့် လွယ်ကူစွာ ပြုလုပ်နိုင်ပါသည်။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. Ruff	https://docs.astral.sh/ruff/	
2. Prettier	https://prettier.io/	
3. စိတ်ကြိုက် ပြင်ဆင်ရန် စက်ဝန်း ကျယ်ပြန့်ပါသည်	https://prettier.io/docs/configuration	
4. version control	https://missing-semester-my.github.io/2026/version-control/	
5. Black	https://github.com/psf/black	
6. gofmt	https://pkg.go.dev/cmd/gofmt	
7. bikeshedding	https://en.wikipedia.org/wiki/Law_of_triviality	
8. IDE integration	https://missing-semester-my.github.io/2026/development-environment/#code-intelligence-and-language-servers	
9. EditorConfig	https://editorconfig.org/	
10. SIM102	https://docs.astral.sh/ruff/rules/collapsible-if/	
11. semgrep	https://github.com/semgrep/semgrep	
12. QuickCheck	https://hackage.haskell.org/package/QuickCheck	
13. Hypothesis	https://hypothesis.readthedocs.io/	
14. Codecov	https://app.codecov.io	
15. pre-commit	https://pre-commit.com/	

Resource / Description	URL	Micro QR
16. hooks	https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks	
17. GitHub Actions	https://github.com/features/actions	
18. semver-compatible	https://missing-semester-my.github.io/2026/shipping-code/#releases--versioning	
19. status badge	https://docs.github.com/en/actions/how-tos/monitor-workflows/add-a-status-badge	
20. proof-html	https://github.com/anishathalye/proof-html	
21. links checker	https://github.com/missing-semester/missing-semester/blob/master/.github/workflows/links.yml	
22. artifact	https://missing-semester-my.github.io/2026/shipping-code/	
23. just	https://github.com/casey/just	
24. npm	https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager	
25. Hatch	https://hatch.pypa.io/	
26. ag	https://github.com/ggreer/the_silver_searcher	
27. go test	https://pkg.go.dev/cmd/go#hdr-Test_packages	
28. Python regex syntax	https://docs.python.org/3/library/re.html	
29. regex101	https://regex101.com/	
30. ရှိသော်လည်း	https://pdw.ex-parrot.com/Mail-RFC822-Address.html	
31. ဤ documentation	https://docs.python.org/3/library/re.html#regular-expression-syntax	

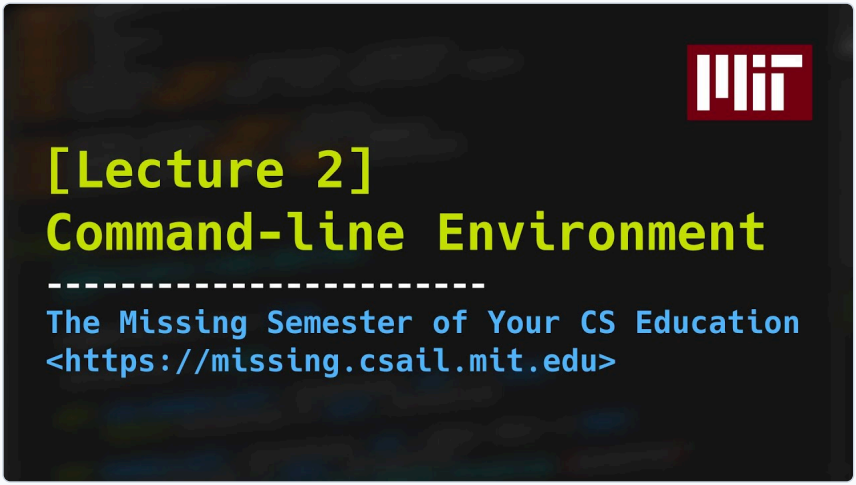
Resource / Description	URL	Micro QR
32. Regular language	https://en.wikipedia.org/wiki/Regular_language	
33. မဖြစ်နိုင်ပါ	https://en.wikipedia.org/wiki/Pumping_lemma_for_regular_languages	
34. pyparsing	https://github.com/pyparsing/pyparsing	
35. PEG	https://en.wikipedia.org/wiki/Parsing_expression_grammar	
36. AI agent	https://missing-semester-my.github.io/2026/agentic-coding/	
37. command-line tool	https://missing-semester-my.github.io/2026/course-shell/	
38. ဤသင်ခန်းစာ မှတ်စုများ	https://raw.githubusercontent.com/missing-semester/missing-semester/refs/heads/master/_2026/code-quality.md	
39. Markdown bullet marker	https://spec.commonmark.org/0.31.2/#bullet-list-marker	

Command-line ပတ်ဝန်းကျင်

အနှစ်ချုပ် - Input/output streams များ၊ environment variables များ နှင့် SSH ဖြင့် remote machine များကို ချိတ်ဆက်အသုံးပြုနည်းများ အပါအဝင် command-line ပရိုဂရမ်များ လုပ်ဆောင်ပုံကို လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Command-line ပတ်ဝန်းကျင်



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=ccBGsPedE9Q

ယခင်သင်ခန်းစာတွင် ဆွေးနွေးခဲ့သည့်အတိုင်း shell အများစုသည် အခြားပရိုဂရမ်များကို စတင်ပွင့်စေသည့် ရိုးရိုး launcher မျှသာမဟုတ်ဘဲ လက်တွေ့တွင် ဘုံသုံး pattern များနှင့် abstraction များ ပြည့်နှက်နေသော ပရိုဂရမ်မင်းဘာသာစကား တစ်ခုလုံးကို ပံ့ပိုးပေးထားပါသည်။ သို့သော် အခြားပရိုဂရမ်မင်းဘာသာစကားအများစုနှင့်မတူဘဲ shell scripting တွင် အရာအားလုံးကို ပရိုဂရမ်များ Run ရန်နှင့် ၎င်းတို့အချင်းချင်း ရိုးရှင်းထိရောက်စွာ ဆက်သွယ်ဆောင်ရွက်နိုင်စေရန်အတွက် ဒီဇိုင်းထုတ်ထားခြင်းဖြစ်ပါသည်။

အထူးသဖြင့် shell scripting သည် *သဘောတူညီချက်များ (conventions)* နှင့် အလွန်နီးကပ်စွာ ဆက်စပ်နေပါသည်။ Command line interface (CLI) ပရိုဂရမ်တစ်ခုသည် ပိုမိုကျယ်ပြန့်သော shell ပတ်ဝန်းကျင်အတွင်း အဆင်ပြေပြေ ပူးပေါင်းလုပ်ဆောင်နိုင်ရန်အတွက် လိုက်နာရန် လိုအပ်သည့် ဘုံ pattern အချို့ ရှိပါသည်။ ယခုအခါ command line ပရိုဂရမ်များ မည်သို့အလုပ်လုပ်သည်ကို နားလည်ရန် လိုအပ်သည့် သဘောတရားအများအပြားနှင့် ၎င်းတို့ကို အသုံးပြုပုံ၊ ချိန်ညှိပြင်ဆင်ပုံဆိုင်ရာ အသုံးများသော သဘောတူညီချက် (conventions) များကို လေ့လာသွားမည် ဖြစ်ပါသည်။

The Command Line Interface

ပရိုဂရမ်မင်းဘာသာစကား အများစုတွင် function တစ်ခု ရေးသားပုံသည် အောက်ပါအတိုင်း ဖြစ်လေ့ရှိသည် -

```
def add(x: int, y: int) -> int:
    return x + y
```

ဒီနေရာမှာ ပရိုဂရမ်ရဲ့ input တွေနဲ့ output တွေကို ရှင်းရှင်းလင်းလင်း မြင်တွေ့နိုင်ပါတယ်။ ၎င်းနှင့် ဆန့်ကျင်ဘက်အနေဖြင့် shell script များကို စတင်ကြည့်ရှုချိန်တွင် တော်တော်လေး ကွဲပြားခြားနားနေသည်ကို တွေ့ရပါလိမ့်မည်။

```
#!/usr/bin/env bash

if [[ -f $1 ]]; then
    echo "Target file already exists"
    exit 1
else
    if $DEBUG; then
        grep 'error' - | tee $1
    else
        grep 'error' - > $1
    fi
    exit 0
fi
```

ယခုကဲ့သို့ script မျိုးတွင် မည်သည့်အရာများ ဖြစ်ပျက်နေသည်ကို သေချာစွာ နားလည်နိုင်ရန်အတွက် shell ပရိုဂရမ်များ အချင်းချင်း သို့မဟုတ် shell ပတ်ဝန်းကျင်နှင့် ဆက်သွယ်ရာတွင် မကြာခဏ ပါဝင်လေ့ရှိသည့် သဘောတရား အနည်းငယ်ကို မိတ်ဆက်ပေးရန် လိုအပ်ပါသည် -

- Arguments များ
- Streams များ
- Environment variables များ
- Return codes များ
- Signals များ

Arguments များ

Shell ပရိုဂရမ်များကို Run သည့်အခါ အဆိုပါ ပရိုဂရမ်များသည် argument စာရင်းတစ်ခုကို လက်ခံရရှိကြပါသည်။ Arguments တွေဟာ shell မှာ ရိုးရိုး string များဖြစ်ကြပြီး ၎င်းတို့ကို မည်သို့ အဓိပ္ပာယ်ဖော်မည်ဆိုသည်မှာ ပရိုဂရမ်အပေါ်၌သာ မူတည်ပါသည်။ ဥပမာအားဖြင့် ကျွန်ုပ်တို့သည် `ls -l folder/` ဟု ရိုက်ထည့်လိုက်သောအခါ `/bin/ls` ပရိုဂရမ်ကို argument များဖြစ်သော `['-l', 'folder/']` တို့ဖြင့် Run ပေးခြင်းဖြစ်ပါသည်။

Shell script အတွင်းမှနေ၍ ၎င်းတို့ကို အထူး shell syntax ဖြင့် ရယူအသုံးပြုနိုင်ပါသည်။ ပထမဆုံး argument ကို ရယူရန် variable `$1` ၊ ဒုတိယ argument အတွက် `$2` စသဖြင့် `$9` အထိ အသုံးပြုနိုင်ပါသည်။ Argument အားလုံးကို စာရင်း (list) အဖြစ် ရယူရန် `$@` ကို သုံးပြီး argument အရေအတွက်ကို ရယူရန် `$#` ကို သုံးပါသည်။ ထို့ပြင် ပရိုဂရမ်၏ အမည်ကိုလည်း `$0` ဖြင့် ရယူနိုင်ပါသည်။

ပရိုဂရမ် အများစုအတွက် argument များသည် *flags* များ နှင့် ရိုးရိုး string များ ရောနှောပါဝင်လေ့ရှိပါသည်။ Flag များကို အရှေ့တွင် dash တိုင်တို (`-`) သို့မဟုတ် double-dash (`--`) ပါဝင်ခြင်းဖြင့် ခွဲခြားသိရှိနိုင်ပါသည်။ Flag များသည် ပုံမှန်အားဖြင့် မထည့်လည်း ရပြီး ၎င်းတို့၏ အဓိကအခန်းကဏ္ဍမှာ ပရိုဂရမ်၏ လုပ်ဆောင်ပုံကို ပြောင်းလဲပေးရန် ဖြစ်ပါသည်။ ဥပမာအားဖြင့် `ls -l` သည် `ls` ၏ output ဖော်ပြပုံပုံစံကို ပြောင်းလဲပေးပါသည်။

သင့်အနေဖြင့် `--all` ကဲ့သို့ နာမည်ရှည်ပါသော double dash flag များနှင့် `-a` ကဲ့သို့ စာလုံးတစ်လုံးတည်းပါလေ့ရှိသော single dash flag များကို တွေ့မြင်ရပါလိမ့်မည်။ ထို option တစ်ခုတည်းကိုပင် ပုံစံနှစ်မျိုးလုံးဖြင့် သတ်မှတ်နိုင်သည်၊ ဥပမာ `ls -a` နှင့် `ls --all` တို့သည် အတူတူပင် ဖြစ်ပါသည်။ Single dash flag များကို စုစည်း၍လည်း ရေးသားနိုင်သဖြင့် `ls -l -a` နှင့် `ls -la` တို့သည်လည်း အတူတူပင် ဖြစ်ပါသည်။ Flag များ၏ အစဉ်လိုက် သတ်မှတ်မှုသည်လည်း ပုံမှန်အားဖြင့် အရေးမကြီးပါ၊ `ls -la` နှင့် `ls -al` တို့သည် ရလဒ်တစ်ခုတည်းကိုပင် ထုတ်ပေးပါသည်။ Flag အချို့သည် အလွန်အသုံးများပြီး shell ပတ်ဝန်းကျင်နှင့် ပိုမိုရင်းနှီးလာသည်နှင့်အမျှ သဘာဝအတိုင်း အလိုလို သုံးစွဲမိလာပါလိမ့်မည် (ဥပမာ `--help` , `--verbose` , `--version`)။

> Flag များသည် shell သဘောတူညီချက်များ (conventions) ၏ ကောင်းမွန်သော ပထမဆုံး ဥပမာ တစ်ခု ဖြစ်ပါသည်။ Shell ဘာသာစကားသည် ကျွန်ုပ်တို့၏ ပရိုဂရမ်အား `-` သို့မဟုတ် `--` ကို ဤကဲ့သို့ သီးသန့် အသုံးပြုရန် အတင်းအကျပ် လိုအပ်ခြင်း မရှိပါ။ `myprogram +myoption myfile` ကဲ့သို့ syntax မျိုးဖြင့် ပရိုဂရမ်တစ်ခု ရေးသားခြင်းကို မည်သည့်အရာကမျှ တားဆီးထားခြင်းမရှိသော်လည်း Dash များကို သုံးစွဲရန် မျှော်လင့်ထားကြသဖြင့် ရှုပ်ထွေးမှုများ ဖြစ်ပေါ်စေပါလိမ့်မည်။ လက်တွေ့တွင် ပရိုဂရမ်မင်းဘာသာစကား အများစုသည် CLI flag parsing library များကို ပံ့ပိုးပေးထားကြပါသည် (ဥပမာ dash syntax ဖြင့် argument များကို parse လုပ်ရန် python ရှိ `argparse`)။

CLI ပရိုဂရမ်များ၏ အခြား အသုံးများသော သဘောတူညီချက်တစ်ခုမှာ ပရိုဂရမ်များသည် အမျိုးအစား တူညီသော argument အရေအတွက် အမျိုးမျိုးကို လက်ခံနိုင်ခြင်း ဖြစ်ပါသည်။ ဤကဲ့သို့ argument များကို ပေးလိုက်သောအခါ command သည် ၎င်းတို့တစ်ခုစီအပေါ် တူညီသော လုပ်ဆောင်ချက်ကို လုပ်ဆောင်ပေးပါသည်။

```
mkdir src
mkdir docs
# သည် အောက်ပါအတိုင်း ရေးသည်နှင့် အတူတူပင်ဖြစ်သည်
mkdir src docs
```

ဤ syntax sugar သည် စတင်ချိန်တွင် မလိုအပ်ဟု ထင်ရနိုင်သော်လည်း *globbing* နှင့် တွဲဖက်လိုက်သောအခါ အလွန်ပင် စွမ်းအားထက်မြက်လာပါသည်။ Globbing သို့မဟုတ် globs တွေဆိုတာ ပရိုဂရမ်ကို မခေါ်မီ shell က အလိုအလျောက် ဖြန့်ကျက်ပေးမည့် (expand) အထူး pattern များ ဖြစ်ကြပါသည်။

ဥပမာ လက်ရှိ folder အတွင်းရှိ .py file အားလုံးကို nonrecursively ဖျက်ပစ်ချင်သည် ဆိုပါစို့။ ယခင် သင်ခန်းစာတွင် လေ့လာခဲ့သည်များအရ အောက်ပါအတိုင်း Run ပြီး ပြုလုပ်နိုင်ပါသည် -

```
for file in $(ls | grep -P '\.py$'); do
    rm "$file"
done
```

သို့သော် ၎င်းကို `rm *.py` ဟု သာ ရေးသားပြီး အစားထိုးနိုင်ပါသည်။

terminal ထဲတွင် `rm *.py` ဟု ရိုက်ထည့်လိုက်သောအခါ shell သည် `/bin/rm` ပရိုဂရမ်ကို `['*.py']` ဆိုသည့် argument ဖြင့် ခေါ်ယူမည် မဟုတ်ပါ။ ယင်းအစား shell သည် လက်ရှိ folder အတွင်း `*.py` pattern နှင့် ကိုက်ညီသော file များကို ရှာဖွေမည်ဖြစ်ပြီး၊ ဤနေရာတွင် `*` သည် မည်သည့်အမျိုးအစားမဆို

ရှိသော စာလုံး အရေအတွက် သုည သို့မဟုတ် သုညထက်ပိုသော string တစ်ခုခုနှင့် ကိုက်ညီမှု ရှိနိုင်ပါသည်။ ထို့ကြောင့် ကျွန်ုပ်တို့၏ folder တွင် `main.py` နှင့် `utils.py` တို့ ရှိနေပါက `rm` ပရိုဂရမ်သည် `['main.py', 'utils.py']` ဆိုသည့် argument များကို လက်ခံရရှိမည် ဖြစ်ပါသည်။

တွေ့ရှိရမည့် အသုံးအများဆုံး glob တွေကတော့ wildcard `*` (သုည သို့မဟုတ် သုညထက်ပိုသော မည်သည့် စာလုံးမဆို)၊ `?` (အတိအကျ တစ်လုံးတည်းသော မည်သည့်စာလုံးမဆို) နှင့် curly braces တွေ ဖြစ်ကြပါတယ်။ Curly braces `{}` သည် ကော်မာခံထားသော pattern စာရင်းကို argument အများအပြားအဖြစ် ဖြန့်ကျက် (expand) ပေးပါသည်။

လက်တွေ့တွင် glob များကို စိတ်ဝင်စားဖွယ် ဥပမာများဖြင့် နားလည်ရလွယ်ကူပါသည် -

```
touch folder/{a,b,c}.py
# အောက်ပါအတိုင်း Expand ဖြစ်သွားမည်
touch folder/a.py folder/b.py folder/c.py

convert image.{png,jpg}
# အောက်ပါအတိုင်း Expand ဖြစ်သွားမည်
convert image.png image.jpg

cp /path/to/project/{setup,build,deploy}.sh /newpath
# အောက်ပါအတိုင်း Expand ဖြစ်သွားမည်
cp /path/to/project/setup.sh /path/to/project/build.sh /path/to/project/deploy.
sh /newpath

# Globbing နည်းလမ်းများကို ပေါင်းစပ်၍လည်း သုံးနိုင်သည်
mv *{.py,.sh} folder
# *.py နှင့် *.sh file အားလုံးကို ရွှေ့ပေးမည် ဖြစ်သည်
```

> အချို့သော shell များ (ဥပမာ zsh) သည် recursive path များအထိ ပါဝင်အောင် expand လုပ်ပေးနိုင်သော `**` ကဲ့သို့ ပိုမိုဆင့်မြင့်သည့် globbing ပုံစံများကိုပင် ပံ့ပိုးပေးထားပါသည်။ ထို့ကြောင့် `rm **/*.py` သည် .py file အားလုံးကို recursive နည်းဖြင့် ဖျက်ပစ်မည် ဖြစ်ပါသည်။

Streams များ

အောက်ပါအတိုင်း ပရိုဂရမ် pipeline တစ်ခုကို Run သည့်အခါတိုင်း -

```
cat myfile | grep -P '\d+' | uniq -c
```

`grep` ပရိုဂရမ်သည် `cat` နှင့် `uniq` ပရိုဂရမ် နှစ်ခုလုံးနှင့် ဆက်သွယ်ဆောင်ရွက်နေသည်ကို တွေ့မြင်နိုင်ပါသည်။

ဤနေရာတွင် သတိပြုရန် အရေးကြီးသော အချက်မှာ ပရိုဂရမ် သုံးခုလုံးသည် တစ်ပြိုင်နက်တည်း Run နေခြင်း ဖြစ်ပါသည်။ အသေးစိတ် ပြောရလျှင် shell သည် ပထမဆုံး cat ကို ခေါ်၊ ပြီးမှ grep ကို ခေါ်၊ ပြီးမှ uniq ကို ခေါ်ခြင်း မဟုတ်ပါ။ ယင်းအစား ပရိုဂရမ် သုံးခုလုံးကို တစ်ပြိုင်နက်တည်း စတင်လိုက်ပြီး shell က cat ၏ output ကို grep ၏ input သို့၊ grep ၏ output ကို uniq ၏ input သို့ ချိတ်ဆက်ပေးခြင်း ဖြစ်ပါသည်။ Pipe operator `|` ကို အသုံးပြုသောအခါ shell သည် စီးကြောင်းတစ်လျှောက်တွင် ပရိုဂရမ်တစ်ခုမှ နောက်တစ်ခုသို့ စီးဆင်းသွားသော data stream များကို စီမံဆောင်ရွက်ပေးပါသည်။

ဤပြိုင်တူလုပ်ဆောင်မှုကို သက်သေပြနိုင်ပါသည်၊ pipeline တစ်ခုအတွင်းရှိ command အားလုံးသည် ချက်ချင်း စတင်ကြပါသည် -

```
$ (sleep 15 && cat numbers.txt) | grep -P '^\d$' | sort | uniq  &
[1] 12345
$ ps | grep -P '(sleep|cat|grep|sort|uniq)'
  32930 pts/1    00:00:00 sleep
  32931 pts/1    00:00:00 grep
  32932 pts/1    00:00:00 sort
  32933 pts/1    00:00:00 uniq
  32948 pts/1    00:00:00 grep
```

`cat` မှလွဲ၍ အခြား process အားလုံးသည် ချက်ချင်း Run နေသည်ကို တွေ့မြင်နိုင်ပါသည်။ Shell သည် ၎င်းတို့ထဲမှ မည်သည့် process မျှ မပြီးဆုံးမီ process အားလုံးကို စတင်ဖွင့်လှစ်ပြီး ၎င်းတို့၏ stream များကို ချိတ်ဆက်ပေးလိုက်ပါသည်။ `cat` သည် sleep ပြီးဆုံးမှသာ စတင်မည်ဖြစ်ပြီး၊ `cat` ၏ output ကို grep ထံ ပို့ပေးကာ ၎င်းမှတစ်ဆင့် ဆက်လက် စီးဆင်းသွားမည် ဖြစ်ပါသည်။

ပရိုဂရမ်တိုင်းတွင် stdin (standard input အတွက်) ဟု ခေါ်သော input stream တစ်ခု ရှိပါသည်။ Pipe လုပ်သောအခါ stdin ကို အလိုအလျောက် ချိတ်ဆက်ပေးပါသည်။ Script အတွင်းတွင် ပရိုဂရမ်အများစုသည် “stdin မှ ဖတ်ရှုမည်” ဟု အဓိပ္ပာယ်ရသော file အမည်အဖြစ် `-` ကို လက်ခံကြပါသည် -

```
# ဒေတာသည် pipe မှ လာသောအခါ ဤနှစ်ခုသည် အတူတူပင်ဖြစ်သည်
echo "hello" | grep "hello"
echo "hello" | grep "hello" -
```

အလားတူပင် ပရိုဂရမ်တိုင်းတွင် output stream နှစ်ခု ရှိကြပါသည် - stdout နှင့် stderr တို့ဖြစ်ကြပါသည်။ Standard output (stdout) သည် အများဆုံး တွေ့ကြုံရသည့် stream ဖြစ်ပြီး ပရိုဂရမ်၏ output ကို pipeline ၏ နောက်ထပ် command ထံသို့ pipe လုပ်ရာတွင် အသုံးပြုပါသည်။ Standard error (stderr) သည် ပရိုဂရမ်များမှ သတိပေးချက်များနှင့် အခြား ပြဿနာအမျိုးအစားများကို သတင်းပို့ရန်အတွက် သီးသန့် ရည်ရွယ်ထားသော တခြား stream တစ်ခုဖြစ်ပြီး အဆိုပါ output သည် စီးကြောင်း၏ နောက် command မှ parse လုပ်ခြင်း မခံရပါ။

```
$ ls /nonexistent
ls: cannot access '/nonexistent': No such file or directory
$ ls /nonexistent | grep "pattern"
ls: cannot access '/nonexistent': No such file or directory
# stderr ကို pipe မလုပ်ထားသောကြောင့် Error စာတမ်းသည် ပေါ်နေဆဲဖြစ်သည်
$ ls /nonexistent 2>/dev/null
# Output မရှိတော့ပါ - stderr ကို /dev/null သို့ လမ်းကြောင်းပြောင်းလိုက်သောကြောင့်ဖြစ်သည်
```

Shell သည် ဤ stream များကို လမ်းကြောင်းပြောင်းရန် (redirect) အတွက် syntax များကို ထောက်ပံ့ပေးထားပါသည်။ အောက်တွင် ဥပမာအချို့ကို ဖော်ပြထားပါသည် -

```
# stdout ကို file တစ်ခုသို့ လမ်းကြောင်းပြောင်းရန် (အသစ်ထပ်ရေးမည်)
echo "hello" > output.txt

# stdout ကို file တစ်ခုသို့ လမ်းကြောင်းပြောင်းရန် (အနောက်မှ ဆက်ရေးမည်)
echo "world" >> output.txt

# stderr ကို file တစ်ခုသို့ လမ်းကြောင်းပြောင်းရန်
ls foobar 2> errors.txt

# stdout နှင့် stderr နှစ်ခုလုံးကို file တစ်ခုတည်းသို့ လမ်းကြောင်းပြောင်းရန်
ls foobar &> all_output.txt

# stdin ကို file တစ်ခုမှ လမ်းကြောင်းပြောင်းယူရန်
grep "pattern" < input.txt

# /dev/null သို့ လမ်းကြောင်းပြောင်း၍ output ကို စွန့်ပစ်ရန်
cmd > /dev/null 2>&1
```

Unix ဒဿန (philosophy) ကို ထင်ဟပ်စေသည့် အခြား စွမ်းအားထက်မြက်သော tool တစ်ခုမှာ fuzzy finder တစ်ခုဖြစ်သည့် `fzf` (https://github.com/junegunn/fzf) ဖြစ်ပါသည်။ ၎င်းသည် stdin မှ စာကြောင်းများကို ဖတ်ရှုပြီး စစ်ထုတ်ရန်နှင့် ရွေးချယ်ရန် အပြန်အလှန်လုပ်ဆောင်နိုင်သော interface တစ်ခုကို ပံ့ပိုးပေးပါသည် -

```
$ ls | fzf
$ cat ~/.bash_history | fzf
```

`fzf` ကို shell လုပ်ဆောင်ချက်များစွာနှင့် ပေါင်းစပ်အသုံးပြုနိုင်ပါသည်။ Shell စိတ်ကြိုက်ပြင်ဆင်ခြင်းအကြောင်း ဆွေးနွေးသည့်အခါ ၎င်း၏ အသုံးပြုပုံများကို ပိုမိုတွေ့မြင်ရပါလိမ့်မည်။

Environment variables များ

Bash တွင် variable များ သတ်မှတ်ရန် `foo=bar` syntax ကို အသုံးပြုပြီး variable ၏ တန်ဖိုးကို ရယူရန် `$foo` syntax ကို သုံးပါသည်။ `foo = bar` ဟု ရေးသားခြင်းသည် မှားယွင်းသော syntax ဖြစ်ကြောင်း သတိပြုပါ၊ အကြောင်းမှာ shell က ၎င်းကို `foo` ပရိုဂရမ်အား `['=', 'bar']` ဆိုသည့် argument များနှင့် ခေါ်ယူခြင်းအဖြစ် parse လုပ်မည်ဖြစ်သောကြောင့် ဖြစ်သည်။ Shell scripting တွင် space (ကွက်လပ်) ၏ အခန်းကဏ္ဍမှာ argument များကို ခွဲခြားပေးရန် ဖြစ်ပါသည်။ ဤပြုမူပုံသည် ရှုပ်ထွေးနိုင်ပြီး ကျင့်သုံးရခက်ခဲနိုင်သဖြင့် စိတ်ထဲတွင် မှတ်သားထားပါ။

Shell variable များတွင် type များ မရှိပါ၊ ၎င်းတို့အားလုံးသည် string များ ဖြစ်ကြသည်။ Shell တွင် string expression များကို ရေးသားသောအခါ single quote (`'`) နှင့် double quote (`"`) တို့သည် အပြန်အလှန် အစားထိုး၍ မရကြောင်း သတိပြုပါ။ `'` ဖြင့် ဝိုင်းထားသော string များသည် စာသားအတိုင်း ဖြစ်ပြီး variable များကို expand လုပ်မည်မဟုတ်၊ command substitution ပြုလုပ်မည်မဟုတ် သို့မဟုတ် escape sequence များကို ပရိုဆက်လုပ်မည်မဟုတ်ပါ။ ၎င်းနှင့် ဆန့်ကျင်ဘက်အနေဖြင့် `"` ဖြင့် ဝိုင်းထားသော string များသည် ထိုသို့ ပြုလုပ်ပေးမည် ဖြစ်သည်။

```
foo=bar
echo "$foo"
# bar ဟု ရိုက်နှိပ်မည်
echo '$foo'
# $foo ဟု ရိုက်နှိပ်မည်
```

Command တစ်ခု၏ output ကို variable တစ်ခုအတွင်း သမ်းဆည်းရန်အတွက် *command substitution* ကို အသုံးပြုပါသည်။ ကျွန်ုပ်တို့သည်

```
files=$(ls)
echo "$files" | grep README
echo "$files" | grep ".py"
```

ဟု Run လိုက်သောအခါ ls ၏ output (အတိအကျပြောရလျှင် stdout) ကို နောက်ပိုင်းတွင် ရယူသုံးစွဲနိုင်သည့် variable `$files` ထဲသို့ ထည့်သွင်းပေးလိုက်မည် ဖြစ်သည်။ `$files` variable ၏ အကြောင်းအရာတွင် ls output မှ newline များ ပါဝင်နေပြီး၊ ၎င်းသည် `grep` ကဲ့သို့သော ပရိုဂရမ်များက item တစ်ခုစီကို သီးခြားစီ လုပ်ဆောင်နိုင်ရန် သိရှိသည့် ပုံစံဖြစ်ပါသည်။

သိသူနည်းပါးသော အလားတူ feature တစ်ခုမှာ *process substitution* ဖြစ်ပြီး၊ `<( CMD )` သည် `CMD` ကို Run ပြီး ရလဒ် output ကို ယာယီ file တစ်ခုအတွင်း ထည့်သွင်းကာ `<()` ကို ထို file ၏ အမည်ဖြင့် အစားထိုးပေးမည် ဖြစ်ပါသည်။ ဒါဟာ command တွေက STDIN အစား file ဖြင့် တန်ဖိုးများ ပေးပို့ရန် မျှော်လင့်ထားသည့်အခါမျိုးတွင် အသုံးဝင်ပါသည်။ ဥပမာအားဖြင့် `diff <(ls src) <(ls docs)` သည် `src` နှင့် `docs` directory နှစ်ခုအတွင်းရှိ file များ၏ ခြားနားချက်များကို ပြသပေးမည် ဖြစ်ပါသည်။

Shell ပရိုဂရမ်တစ်ခုသည် အခြားပရိုဂရမ်တစ်ခုကို ခေါ်ယူသည့်အခါတိုင်း ၎င်းသည် *environment variables* ဟု အလွယ်ခေါ်ကြသော variable အစုအဝေးတစ်ခုကို ထပ်ဆင့် ပေးပို့ပေးပါသည်။ Shell အတွင်းမှနေ၍ လက်ရှိ environment variable များကို `printenv` ကို Run ခြင်းဖြင့် ရှာဖွေနိုင်ပါသည်။ Environment variable တစ်ခုကို အတိအကျ ပေးပို့ရန်အတွက် command ၏ အရှေ့တွင် variable သတ်မှတ်ချက်ကို အောက်ပါအတိုင်း ရှေ့ဆက်ထည့်နိုင်သည် -

> Environment variable များကို သဘောတူညီချက် (convention) အဖြစ် စာလုံးကြီးများ (ALL_CAPS) ဖြင့် ရေးသားလေ့ရှိကြပါသည် (ဥပမာ `HOME`, `PATH`, `DEBUG`)။ ဒါဟာ နည်းပညာအရ မဖြစ်မနေ လိုအပ်ချက်မဟုတ်ဘဲ convention တစ်ခုသာ ဖြစ်သော်လည်း ၎င်းကို လိုက်နာခြင်းဖြင့် ပုံမှန်အားဖြင့် စာလုံးသေးဖြင့် ရေးသားသော local shell variable များနှင့် environment variable များကို ခွဲခြားသိမြင်စေရန် ကူညီပေးပါသည်။

```
TZ=Asia/Tokyo date  # တိုကျိုမြို့၏ လက်ရှိအချိန်ကို ရိုက်နှိပ်မည်
echo $TZ  # TZ ကို child command အတွက်သာ သတ်မှတ်ခဲ့သောကြောင့် ဤနေရာတွင် လွတ်နေမည် ဖြစ်သည်
```

သို့မဟုတ်ပါက ကျွန်ုပ်တို့၏ လက်ရှိ environment ကို ပြုပြင်ပြောင်းလဲပေးမည့် `export` built-in function ကို အသုံးပြုနိုင်ပြီး ထိုသို့ ပြုလုပ်ခြင်းဖြင့် child process အားလုံးသည် အဆိုပါ variable ကို လက်ခံရရှိသွားမည် ဖြစ်သည် -

```
export DEBUG=1
# ဤအချိန်မှစ၍ ပရိုဂရမ်အားလုံးသည် ၎င်းတို့၏ environment တွင် DEBUG=1 ပါရှိသွားမည် ဖြစ်သည်
bash -c 'echo $DEBUG'
# 1 ဟု ရိုက်နှိပ်မည်
```

Variable တစ်ခုကို ဖျက်ပစ်ရန်အတွက် `unset` built-in command ကို သုံးပါ၊ ဥပမာ `unset DEBUG` ။

> Environment variable တွေဟာ အခြားသော shell သဘောတူညီချက် (convention) တစ်ခု ဖြစ်ပါတယ်။ ၎င်းတို့ကို ပရိုဂရမ်အများအပြား၏ ပြုမူပုံကို အတိအကျ ပြင်ဆင်ခိုင်းခြင်းထက် သွယ်ဝိုက်သော နည်းလမ်းဖြင့် ပြောင်းလဲရန် အသုံးပြုနိုင်ပါသည်။ ဥပမာအားဖြင့် shell သည် လက်ရှိ user ၏ home folder path ဖြင့် `$HOME` environment variable ကို သတ်မှတ်ပေးထားပါသည်။ ထို့ကြောင့် ပရိုဂရမ်များသည် `--home /home/alice` ကဲ့သို့ အတိအကျ တောင်းဆိုရန် မလိုဘဲ ဤအချက်အလက်ကို ရယူရန် အဆိုပါ variable ကို အသုံးပြုနိုင်ပါသည်။ အခြား အသုံးများသော ဥပမာ တစ်ခုမှာ `$TZ` ဖြစ်ပြီး ပရိုဂရမ်များစွာသည် သတ်မှတ်ထားသော စံတော်ချိန် (timezone) အလိုက် ရက်စွဲနှင့် အချိန်များကို ပုံစံထုတ်ရန် သုံးစွဲကြပါသည်။

Return codes များ

ယခင်က တွေ့မြင်ခဲ့ရသည့်အတိုင်း shell ပရိုဂရမ်တစ်ခု၏ အဓိက output ကို stdout/stderr stream များနှင့် filesystem ဘေးထွက်ဆိုးကျိုး (side effects) များမှတစ်ဆင့် ဖော်ပြပေးပါသည်။

မူလအားဖြင့် shell script တစ်ခုသည် exit code သုည (0) ကို Return ပြန်ပေးမည် ဖြစ်ပါသည်။ သဘောတူညီချက် (convention) အရ သုညသည် အရာအားလုံး အဆင်ပြေပြေ ပြီးမြောက်ခဲ့သည်ဟု အဓိပ္ပာယ်ရပြီး သုညမဟုတ်သော တန်ဖိုး (nonzero) သည် ပြဿနာအချို့ ကြုံတွေ့ခဲ့ရသည်ဟု အဓိပ္ပာယ်ရပါသည်။ သုညမဟုတ်သော exit code ကို Return ပြန်ပေးရန်အတွက် `exit NUM` ဆိုသည့် shell built-in ကို အသုံးပြုရပါမည်။ နောက်ဆုံး Run ခဲ့သော command ၏ return code ကို အထူး variable `$?` ကို ဝင်ရောက်ကြည့်ရှုခြင်းဖြင့် ရယူနိုင်ပါသည်။

Shell တွင် AND နှင့် OR လုပ်ဆောင်ချက်များကို လုပ်ဆောင်ရန်အတွက် boolean operator များဖြစ်ကြသော `&&` နှင့် `||` တို့ ပါရှိပါသည်။ ပုံမှန် ပရိုဂရမ်မင်းဘာသာစကားများတွင် တွေ့ရသည်များနှင့် မတူဘဲ shell ရှိ operator များသည် ပရိုဂရမ်များ၏ return code အပေါ် မူတည်၍ လုပ်ဆောင်ကြခြင်း ဖြစ်ပါသည်။ ဤနှစ်ခုလုံးသည် **short-circuiting** (https://en.wikipedia.org/wiki/Short-circuit_evaluation) operator များ ဖြစ်ကြပါသည်။ ဆိုလိုသည်မှာ ယခင် command များ၏ အောင်မြင်မှု သို့မဟုတ် လှဲမှားမှုအပေါ် မူတည်၍ နောက် command

များကို အခြေအနေအလိုက် Run ရန် ၎င်းတို့ကို အသုံးပြုနိုင်ပြီး၊ အောင်မြင်မှုကို return code သည် သုည ဟုတ်မဟုတ်အပေါ် မူတည်၍ ဆုံးဖြတ်ခြင်း ဖြစ်ပါသည်။ ဥပမာအချို့ -

```
# grep အောင်မြင်ပါက (ကိုက်ညီမှုတွေ့ရှိပါက) မှသာ echo ကို Run မည်ဖြစ်သည်
grep -q "pattern" file.txt && echo "Pattern found"

# grep မအောင်မြင်ပါက (ကိုက်ညီမှုမတွေ့ပါက) မှသာ echo ကို Run မည်ဖြစ်သည်
grep -q "pattern" file.txt || echo "Pattern not found"

# true သည် အမြဲတမ်း အောင်မြင်သော shell ပရိုဂရမ်တစ်ခု ဖြစ်သည်
true && echo "This will always print"

# false သည် အမြဲတမ်း မအောင်မြင်သော shell ပရိုဂရမ်တစ်ခု ဖြစ်သည်
false || echo "This will always print"
```

အလားတူ သဘောတရားကို `if` နှင့် `while` statement များတွင်လည်း ကျင့်သုံးသည်၊ ၎င်းတို့ နှစ်ခုလုံးသည် ဆုံးဖြတ်ချက်ချရန်အတွက် return code များကို အသုံးပြုကြပါသည် -

```
# if သည် condition command ၏ return code ကို အသုံးပြုသည် (0 = true, nonzero = fa
lse)
if grep -q "pattern" file.txt; then
    echo "Found"
fi

# while loop သည် command က 0 ပြန်ပေးနေသရွေ့ ဆက်လက် လုပ်ဆောင်နေမည်ဖြစ်သည်
while read line; do
    echo "$line"
done < file.txt
```

Signals များ

အချို့သော အခြေအနေများတွင် ပရိုဂရမ်တစ်ခု လုပ်ဆောင်နေစဉ်အတွင်း ဥပမာ command တစ်ခု ပြီးမြောက်ရန် အချိန်ကြာလွန်းနေပါက ၎င်းအား ကြားဖြတ်ရပ်တန့်ရန် လိုအပ်ပါလိမ့်မည်။ ပရိုဂရမ်တစ်ခုကို ကြားဖြတ်ရပ်တန့်ရန် အရိုးရှင်းဆုံး နည်းလမ်းမှာ `Ctrl-C` ကို နှိပ်လိုက်ခြင်းဖြစ်ပြီး အဆိုပါ command သည် ရပ်တန့်သွားလေ့ရှိပါသည်။ သို့သော် ဒါဟာ လက်တွေ့မှာ မည်သို့ အလုပ်လုပ်သနည်း၊ ၎င်းသည် ရုံဖန်ရုံခါ process ကို အဘယ်ကြောင့် ရပ်တန့်ရန် ပျက်ကွက်ရသနည်း။

```
$ sleep 100
^C
$
```

> ဤနေရာတွင် `^C` ဆိုသည်မှာ terminal ထဲတွင် `Ctrl-C` ကို ရိုက်ထည့်လိုက်သောအခါ ဖော်ပြပေးသည့် ပုံစံဖြစ်ပါသည်။

ကွယ်လွန်နောက်ကွယ်တွင် ဖြစ်ပျက်သွားသည်မှာ အောက်ပါအတိုင်း ဖြစ်ပါသည် -

1. ကျွန်ုပ်တို့ `Ctrl-C` ကို နှိပ်လိုက်သည်
2. Shell က ထိုအထူး စာလုံးပေါင်းစပ်မှုကို ခွဲခြားသိရှိလိုက်သည်
3. Shell process က SIGINT signal တစ်ခုကို `sleep` process ထံသို့ ပေးပို့လိုက်သည်
4. Signal သည် `sleep` process ၏ လုပ်ဆောင်မှုကို ကြားဖြတ်ရပ်တန့်လိုက်သည်

Signal များသည် အထူး ဆက်သွယ်ရေး ယန္တရားတစ်ခု ဖြစ်ကြသည်။ Process တစ်ခုသည် signal တစ်ခုကို လက်ခံရရှိသောအခါ ၎င်း၏ လုပ်ဆောင်မှုကို ရပ်တန့်လိုက်ပြီး signal ကို ဖြေရှင်းကာ အဆိုပါ signal မှ ပေးပို့လိုက်သော အချက်အလက်များအပေါ် မူတည်၍ လုပ်ဆောင်မှု စီးဆင်းပုံကို ပြောင်းလဲနိုင်ပါသည်။ ဤအကြောင်းကြောင့် signal များကို *software interrupts* ဟု ခေါ်ဆိုကြပါသည်။

ကျွန်ုပ်တို့၏ အခြေအနေတွင် `Ctrl-C` ကို ရိုက်ထည့်လိုက်သောအခါ ၎င်းသည် process ထံသို့ `SIGINT` signal ပေးပို့ရန် shell ကို တိုက်တွန်းလိုက်ခြင်း ဖြစ်ပါသည်။ အောက်တွင် `SIGINT` ကို ဖမ်းယူပြီး ပစ်ပယ်ထားကာ ဆက်လက် ရပ်တန့်ခြင်းမရှိတော့သည့် Python ပရိုဂရမ်၏ အသေးငယ်ဆုံး ဥပမာကို ဖော်ပြထားပါသည်။ ဤပရိုဂရမ်ကို Kill လုပ်ရန်အတွက် `Ctrl-\` ကို ရိုက်ထည့်ခြင်းဖြင့် `SIGQUIT` signal ကို အစားထိုး အသုံးပြုနိုင်ပါပြီ။

```
#!/usr/bin/env bash
import signal, time

def handler(signum, time):
    print("\nI got a SIGINT, but I am not stopping")

signal.signal(signal.SIGINT, handler)
i = 0
while True:
    time.sleep(.1)
    print("\r{}".format(i), end="")
    i += 1
```

ဤပရိုဂရမ်ထံသို့ `SIGINT` နှစ်ကြိမ် ပေးပို့ပြီးနောက် `SIGQUIT` ပေးပို့လိုက်သောအခါ ဖြစ်ပေါ်လာပုံမှာ အောက်ပါအတိုင်း ဖြစ်ပါသည်။ Terminal ထဲတွင် ရိုက်ထည့်လိုက်သောအခါ `Ctrl` ကို `^` ဖြင့် ဖော်ပြကြောင်း သတိပြုပါ -

```
$ python sigint.py
24^C
I got a SIGINT, but I am not stopping
26^C
I got a SIGINT, but I am not stopping
30^\[1]    39913 quit       python sigint.py
```

`SIGINT` နှင့် `SIGQUIT` နှစ်ခုလုံးသည် ပုံမှန်အားဖြင့် terminal ဆိုင်ရာ တောင်းဆိုချက်များနှင့် ဆက်စပ်နေသော်လည်း၊ process တစ်ခုအား သပ်သပ်ရပ်ရပ် ပိတ်သိမ်းရန် တောင်းဆိုသည့် ပိုမို အထွေထွေကျသော signal မှာ `SIGTERM` signal ဖြစ်ပါသည်။ ဤ signal ကို ပေးပို့ရန်အတွက် **`kill`** (https://www.man7.org/linux/man-pages/man1/kill.1.html) command ကို `kill -TERM <PID>` syntax ဖြင့် အသုံးပြုနိုင်ပါသည်။

Signal များသည် process တစ်ခုကို kill ပြုလုပ်ခြင်းထက် အခြားအရာများကိုလည်း ဆောင်ရွက်နိုင်ပါသည်။ ဥပမာ `SIGSTOP` သည် process ကို ခဏရပ်တန့် (pause) စေပါသည်။ Terminal တွင် `Ctrl-Z` ရိုက်ထည့်ပါက Terminal Stop ၏ အတိုကောက်ဖြစ်သော `SIGTSTP` signal ကို ပေးပို့ရန် shell ကို တိုက်တွန်းမည်ဖြစ်သည် (ဆိုလိုသည်မှာ `SIGSTOP` ၏ terminal ဗားရှင်းဖြစ်သည်)။

ထို့နောက် ရပ်တန့်ထားသော job ကို foreground တွင် ဖြစ်စေ၊ background တွင် ဖြစ်စေ အသီးသီး **`fg`** (https://www.man7.org/linux/man-pages/man1/fg.1p.html) သို့မဟုတ် **`bg`** (https://man7.org/linux/man-pages/man1/bg.1p.html) များကို အသုံးပြု၍ ဆက်လက် လုပ်ဆောင်စေနိုင်ပါသည်။

`jobs` (https://www.man7.org/linux/man-pages/man1/jobs.1p.html) command သည် လက်ရှိ terminal session နှင့် ဆက်စပ်နေသော မပြီးဆုံးသေးသည့် job များကို စာရင်းဖော်ပြပေးပါသည်။ အဆိုပါ job များကို ၎င်းတို့၏ pid ကို အသုံးပြု၍ ညွှန်းဆိုနိုင်သည် (ရှာဖွေရန် `pgrep` (https://www.man7.org/linux/man-pages/man1/pgrep.1.html) ကို သုံးနိုင်သည်)။ ပိုမိုလွယ်ကူစွာဖြင့် ၎င်းတို့၏ job number (`jobs` မှ ဖော်ပြထားသော) ရှေ့တွင် ရာခိုင်နှုန်းသင်္ကေတ (%) ကို တပ်၍လည်း ညွှန်းဆိုနိုင်ပါသည်။ နောက်ဆုံး background သို့ ပို့ထားသော job ကို ညွှန်းဆိုရန် အထူး parameter ဖြစ်သော `$!` ကို အသုံးပြုနိုင်ပါသည်။

သိရှိထားရန် အခြားအချက်တစ်ခုမှာ command တစ်ခု၏ အနောက်တွင် `&` ထည့်သွင်းလိုက်ပါက command ကို background တွင် Run ပေးမည်ဖြစ်ပြီး သင့်ထံ prompt ပြန်လည် ရရှိစေမည် ဖြစ်သည်၊ သို့သော် ၎င်းသည် shell ၏ STDOUT ကို သုံးနေဆဲဖြစ်သဖြင့် စိတ်အနှောင့်အယှက် ဖြစ်စရာ ကောင်းနိုင်ပါသည် (ထိုသို့ဖြစ်ပါက shell redirection များကို သုံးပါ)။ အလားတူပင် Run နေပြီးသား ပရိုဂရမ်တစ်ခုကို background သို့ ပို့ရန် `Ctrl-Z` နှိပ်ပြီးနောက် `bg` ဟု ရိုက်ထည့်နိုင်ပါသည်။

Background သို့ ပို့ထားသော process များသည် သင့် terminal ၏ child process များဖြစ်နေဆဲဖြစ်ပြီး terminal ကို ပိတ်လိုက်ပါက သေဆုံးသွားမည်ဖြစ်ကြောင်း သတိပြုပါ (ဒါဟာ အခြား signal တစ်ခုဖြစ်သည့် `SIGHUP` ကို ပေးပို့လိုက်ခြင်း ဖြစ်သည်)။ ထိုသို့ မဖြစ်ပွားစေရန်အတွက် ပရိုဂရမ်ကို `nohup` (https://www.man7.org/linux/man-pages/man1/nohup.1.html) (`SIGHUP` ကို ပစ်ပယ်ရန် wrapper) ဖြင့် Run နိုင်သည်၊ သို့မဟုတ် process က စတင်နေပြီးပါက `disown` ကို အသုံးပြုနိုင်ပါသည်။ သို့မဟုတ်ပါက နောက်အပိုင်းတွင် လေ့လာရမည့် terminal multiplexer တစ်ခုကို အသုံးပြုနိုင်ပါသည်။

အောက်တွင် ဤသဘောတရားအချို့ကို ပြသရန် နမူနာ session တစ်ခုကို ဖော်ပြထားပါသည် -

```
$ sleep 1000
^Z
[1]  + 18653 suspended  sleep 1000

$ nohup sleep 2000 &
[2] 18745
appending output to nohup.out

$ jobs
[1]  + suspended  sleep 1000
[2]  - running    nohup sleep 2000

$ kill -SIGHUP %1
[1]  + 18653 hangup     sleep 1000

$ kill -SIGHUP %2   # nohup protects from SIGHUP

$ jobs
[2]  + running    nohup sleep 2000

$ kill %2
[2]  + 18745 terminated  nohup sleep 2000
```

`SIGKILL` သည် ထူးခြားသော signal တစ်ခုဖြစ်သည်၊ အကြောင်းမှာ ၎င်းကို process မှ ဖမ်းယူထား၍ မရနိုင်ဘဲ ချက်ချင်း အမြဲတမ်း ပြီးဆုံးစေသောကြောင့် ဖြစ်သည်။ သို့သော် ၎င်းတွင် မိဘမဲ့ child process များ ကျန်ရစ်ခဲ့ခြင်းကဲ့သို့သော မကောင်းသော ဘေးထွက်ဆိုးကျိုးများ ရှိနိုင်ပါသည်။

ဤအရာများနှင့် အခြား signal များအကြောင်းကို [ဤနေရာတွင်](https://en.wikipedia.org/wiki/Signal_(IPC)) သို့မဟုတ် [`man signal`](https://www.man7.org/linux/man-pages/man1/signal.7.html) သို့မဟုတ် `kill -l` ဟု ရိုက်ထည့်၍ ပိုမို လေ့လာနိုင်ပါသည်။

Shell script များအတွင်း Signal များကို လက်ခံရရှိသည့်အခါ command များကို စီမံဆောင်ရွက်ရန် `trap` built-in ကို သုံးနိုင်သည်။ ဒါဟာ ရှင်းလင်းရေး လုပ်ငန်းစဉ်များ (cleanup) အတွက် အလွန် အသုံးဝင်ပါသည် -

```
#!/usr/bin/env bash
cleanup() {
    echo "Cleaning up temporary files..."
    rm -f /tmp/mytemp.*
}
trap cleanup EXIT  # Script မှ ထွက်ခွာသည့်အခါ cleanup ကို Run မည်
trap cleanup SIGINT SIGTERM  # Ctrl-C သို့မဟုတ် kill ပြုလုပ်သည့်အခါတွင်လည်း Run မည်
```

Remote Machine များ

ပရိုဂရမာများအနေဖြင့် ၎င်းတို့၏ နေ့စဉ်အလုပ်တွင် remote server များနှင့် တွဲဖက်လုပ်ဆောင်ရခြင်းမှာ ပိုမိုခေတ်စားလာပါသည်။ ဤနေရာတွင် အလုပ်အတွက် အသုံးအများဆုံး tool မှာ SSH (Secure Shell) ဖြစ်ပြီး ၎င်းသည် remote server သို့ ချိတ်ဆက်နိုင်ရန်နှင့် ယခု ရင်းနှီးနေပြီဖြစ်သော shell interface ကို ပံ့ပိုးပေးနိုင်ရန် ကူညီပေးပါလိမ့်မည်။ ကျွန်ုပ်တို့သည် server တစ်ခုသို့ အောက်ပါအတိုင်း command ဖြင့် ချိတ်ဆက်ကြပါသည် -

```
ssh alice@server.mit.edu
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် `server.mit.edu` server ရှိ user `alice` အဖြစ် ssh ဝင်ရောက်ရန် ကြိုးပမ်းနေခြင်း ဖြစ်ပါသည်။

`ssh` ၏ သတိမမူမိတတ်ကြသော feature တစ်ခုမှာ command များကို non-interactively (အပြန်အလှန်တုံ့ပြန်စရာ မလိုဘဲ) Run နိုင်သည့် စွမ်းရည် ဖြစ်ပါသည်။ `ssh` သည် command ၏ stdin ပေးပို့ခြင်းနှင့် stdout လက်ခံရရှိခြင်းတို့ကို မှန်ကန်စွာ စီမံပေးသဖြင့် ၎င်းကို အခြား command များနှင့် ပေါင်းစပ်အသုံးပြုနိုင်ပါသည် -

```
# ဤနေရာတွင် ls ကို remote တွင် Run ပြီး wc ကို local တွင် Run သည်
ssh alice@server ls | wc -l

# ဤနေရာတွင် ls နှင့် wc နှစ်ခုလုံးကို server တွင် Run သည်
ssh alice@server 'ls | wc -l'
```

> ချိတ်ဆက်မှုပြတ်တောက်ခြင်း၊ sleep ဝင်/ထွက်ခြင်း၊ ကွန်ရက်ပြောင်းလဲခြင်းနှင့် latency မြင့်မားသော ချိတ်ဆက်မှုများကို စီမံနိုင်သည့် SSH အစားထိုး tool အဖြစ် **Mosh** (https://mosh.org/) ကို install လုပ်ကြည့်ပါ။

`ssh` က ကျွန်ုပ်တို့အား remote server တွင် command များကို Run ခွင့်ပြုရန်အတွက် ကျွန်ုပ်တို့သည် ထိုသို့ပြုလုပ်ရန် အခွင့်အာဏာရှိကြောင်း သက်သေပြရန် လိုအပ်ပါသည်။ ၎င်းကို password များ သို့မဟုတ် ssh key များမှတစ်ဆင့် ပြုလုပ်နိုင်ပါသည်။ Key အခြေပြု authentication သည် အဓိက လျှို့ဝှက် private key

ကို ထုတ်ဖော်ပြသခြင်း မရှိဘဲ client သည် ထို key ကို ပိုင်ဆိုင်ကြောင်း server သို့ သက်သေပြရန်အတွက် public-key cryptography ကို အသုံးပြုပါသည်။ Key အခြေပြု authentication သည် ပိုမို အဆင်ပြေပြီး ပိုမို လုံခြုံသဖြင့် ၎င်းကိုသာ ဦးစားပေး အသုံးပြုသင့်ပါသည်။ Private key (မကြာခဏ `~/.ssh/id_rsa` နှင့် နောက်ပိုင်းတွင် `~/.ssh/id_ed25519`) သည် ထိရောက်စွာဖြင့် သင့် password ဖြစ်သောကြောင့် ၎င်း အတိုင်း သဘောထားပါ၊ ၎င်း၏ အကြောင်းအရာများကို မည်သူ့ကိုမျှ မျှဝေခြင်း မပြုပါနှင့်။

Key တွဲတစ်ခုကို ထုတ်လုပ်ရန်အတွက် **ssh-keygen** (https://www.man7.org/linux/man-pages/man1/ssh-keygen.1.html) ကို Run နိုင်ပါသည်။

```
ssh-keygen -a 100 -t ed25519 -f ~/.ssh/id_ed25519
```

အကယ်၍ သင့်အနေဖြင့် SSH key များ သုံးပြီး GitHub သို့ push ပြုလုပ်ရန် ချိန်ညှိဖူးပါက ဤနေရာတွင် (https://help.github.com/articles/connecting-to-github-with-ssh/) ဖော်ပြထားသော အဆင့်များကို ပြုလုပ်ဖူးပြီးသားဖြစ်နိုင်ပြီး တရားဝင် key တွဲတစ်ခု ရှိနှင့်ပြီး ဖြစ်ပါလိမ့်မည်။ သင့်တွင် passphrase ရှိမရှိ စစ်ဆေးရန်နှင့် အတည်ပြုရန်အတွက် `ssh-keygen -y -f /path/to/key` ကို Run နိုင်ပါသည်။

Server ဘက်တွင် `ssh` သည် မည်သည့် client များကို ဝင်ရောက်ခွင့်ပြုရမည်နည်းဆိုသည်ကို ဆုံးဖြတ်ရန် `.ssh/authorized_keys` တွင် ကြည့်ရှုမည် ဖြစ်ပါသည်။ Public key တစ်ခုကို ကူးယူထည့်သွင်းရန် အောက်ပါအတိုင်း သုံးနိုင်သည် -

```
cat .ssh/id_ed25519.pub | ssh alice@remote 'cat >> ~/.ssh/authorized_keys'

# သို့မဟုတ် ပိုမို ရိုးရှင်းစွာဖြင့် (ssh-copy-id ရှိပါက)

ssh-copy-id -i .ssh/id_ed25519 alice@remote
```

Command များကို Run ခြင်းအပြင် SSH ဖြင့် ထူထောင်ထားသော ချိတ်ဆက်မှုကို server ထံသို့ သို့မဟုတ် server ထံမှ file များကို လုံခြုံစွာ လွှဲပြောင်းရန်လည်း အသုံးပြုနိုင်ပါသည်။ **scp** (https://www.man7.org/linux/man-pages/man1/scp.1.html) သည် ရိုးရာအကျဆုံး tool ဖြစ်ပြီး syntax မှာ `scp path/to/local_file remote_host:path/to/remote_file` ဖြစ်ပါသည်။ **rsync** (https://www.man7.org/linux/man-pages/man1/rsync.1.html) သည် local နှင့် remote တွင် တူညီသော file များကို ရှာဖွေ တွေ့ရှိခြင်းဖြင့် ၎င်းတို့ကို ထပ်မံ ကူးယူခြင်းမှ တားဆီးကာ `scp` ထက် ပိုမို ကောင်းမွန်အောင် ပြုလုပ်ထားပါသည်။ ၎င်းသည် symlink များနှင့် permission များအပေါ် ပိုမို သီးသန့် ထိန်းချုပ်နိုင်မှုကို ပေးစွမ်းပြီး ယခင် ကြားဖြတ် ရပ်တန့်သွားသော copy ပြုလုပ်မှုကို ပြန်လည် စတင်နိုင်သည့် `--partial` flag ကဲ့သို့သော အပို feature များ ပါရှိပါသည်။ `rsync` တွင် `scp` နှင့် အလားတူ syntax ရှိပါသည်။

SSH client ချိန်ညှိချက်များသည် `~/.ssh/config` တွင် တည်ရှိပြီး ၎င်းသည် host များကို ကြေညာရန်နှင့် ၎င်းတို့အတွက် မူလ setting များကို သတ်မှတ်ရန် ခွင့်ပြုပေးပါသည်။ ဤ configuration file ကို `ssh` ကသာ မက `scp`, `rsync`, `mosh` အစရှိသည့် အခြား ပရိုဂရမ်များကလည်း ဖတ်ရှုကြပါသည်။

```
Host vm
    User alice
    HostName 172.16.174.141
    Port 2222
    IdentityFile ~/.ssh/id_ed25519

# Configuration များတွင် wildcard များကိုလည်း သုံးနိုင်သည်
Host *.mit.edu
    User alice
```

Terminal Multiplexers

Command line interface ကို အသုံးပြုသည့်အခါ အရာတစ်ခုထက်မက တစ်ပြိုင်နက်တည်း Run လိုသည့် အခြေအနေများ မကြာခဏ ရှိပါလိမ့်မည်။ ဥပမာ သင့်အနေဖြင့် သင့် editor နှင့် သင့်ပရိုဂရမ်ကို ဘေးချင်းယှဉ် Run လိုနိုင်ပါသည်။ ၎င်းကို terminal window အသစ်များ ဖွင့်လှစ်ခြင်းဖြင့် အောင်မြင်နိုင်သော်လည်း terminal multiplexer တစ်ခုကို အသုံးပြုခြင်းသည် ပိုမို စွမ်းဆောင်ရည်စုံလင်သော ဖြေရှင်းချက်တစ်ခု ဖြစ်ပါသည်။

`tmux` (https://www.man7.org/linux/man-pages/man1/tmux.1.html) ကဲ့သို့သော Terminal multiplexer များသည် ပိုမို ထိရောက်သော နည်းလမ်းဖြင့် shell session အများအပြားနှင့် ဆက်သွယ်နိုင်စေရန် pane များနှင့် tab များကို အသုံးပြု၍ terminal window များကို multiplex ပြုလုပ်ခွင့် ပေးထားပါသည်။ ထို့ပြင် terminal multiplexer များသည် လက်ရှိ terminal session ကို ဖြုတ်ထုတ်ထားနိုင်ခဲ့ (detach) ပြီး နောက်ပိုင်းတွင် ပြန်လည် ချိတ်ဆက်နိုင်ခဲ့ (reattach) ပါသည်။ ဤအကြောင်းကြောင့် terminal multiplexer များသည် `nohup` သို့မဟုတ် အလားတူ လှည့်ကွက်များကို သုံးရန် မလိုတော့ဘဲ remote machine များနှင့် အလုပ်လုပ်သည့်အခါ အလွန်ပင် အဆင်ပြေလှပါသည်။

ယခုခေတ်တွင် အသုံးအများဆုံး terminal multiplexer မှာ `tmux` (https://www.man7.org/linux/man-pages/man1/tmux.1.html) ဖြစ်ပါသည်။ `tmux` သည် စိတ်ကြိုက်ပြင်ဆင်နိုင်စွမ်း မြင့်မားပြီး သက်ဆိုင်ရာ keybinding များကို အသုံးပြု၍ tab များနှင့် pane အများအပြားကို ဖန်တီးကာ ၎င်းတို့အကြား လျင်မြန်စွာ သွားလာနိုင်ပါသည်။

`tmux` သည် သင့်အား ၎င်း၏ keybinding များကို သိရှိထားရန် မျှော်လင့်ထားပြီး၊ ၎င်းတို့အားလုံးသည် `<C-b> x` ပုံစံ ရှိကြပြီး ဆိုလိုသည်မှာ (၁) `Ctrl+b` ကို နှိပ်ပါ၊ (၂) `Ctrl+b` ကို လွှတ်ပါ၊ ထို့နောက် (၃) `x` ကို နှိပ်ပါ။ `tmux` တွင် အောက်ပါ အဆင့်ဆင့် ပါဝင်သော အရာများ (hierarchy of objects) ရှိပါသည် - - **Sessions** - session တစ်ခုသည် window တစ်ခု သို့မဟုတ် တစ်ခုထက်မက ပါဝင်သော သီးခြား workspace တစ်ခု ဖြစ်သည်

```
+ `tmux` သည် session အသစ်တစ်ခုကို စတင်ပေးသည်
+ `tmux new -s NAME` သည် ၎င်းအမည်ဖြင့် စတင်ပေးသည်
+ `tmux ls` သည် လက်ရှိ session များကို စာရင်းဇော်ပြပေးသည်
+ `tmux` အတွင်း `<C-b> d` ရိုက်ထည့်ပါက လက်ရှိ session ကို detach ပြုလုပ်ပေးသည်
+ `tmux a` သည် နောက်ဆုံး session ကို attach ပြုလုပ်ပေးသည်။ မည်သည့် session ကို မည်သို့ သတ်မှတ်ရန် `-t` flag ကို သုံးနိုင်သည်
```

- **Windows** - Editor သို့မဟုတ် browser ရှိ tab များနှင့် တူညီပြီး၊ ၎င်းတို့သည် session တစ်ခုတည်း၏ အမြင်အရ သီးခြား ဖြစ်နေသော အစိတ်အပိုင်းများ ဖြစ်ကြသည်

```
+ `<C-b> c` Window အသစ်တစ်ခု ဖန်တီးပေးသည်။ ၎င်းကို ပိတ်ရန် `<C-d>` ပြုလုပ်ပြီး shell များကို ပိတ်သိမ်းနိုင်သည်
+ `<C-b> N` အမှတ်စဉ် _N_ မြောက် window သို့ သွားမည်။ ၎င်းတို့ကို နံပါတ် တပ်ထားကြောင်း သတိပြုပါ
+ `<C-b> p` ယခင် window သို့ သွားမည်
+ `<C-b> n` နောက် window သို့ သွားမည်
+ `<C-b> ,` လက်ရှိ window ၏ အမည်ကို ပြောင်းလဲမည်
+ `<C-b> w` လက်ရှိ window များကို စာရင်းဖော်ပြမည်
```

- **Panes** - Vim split များနှင့် တူပြီး၊ pane များသည် မျက်နှာပြင် အမြင်တစ်ခုတည်းတွင် shell အများအပြားကို ယှဉ်တွဲ ရှိစေနိုင်သည်

```
+ `<C-b> "` လက်ရှိ pane ကို အလျားလိုက် ပိုင်းခြားမည်
+ `<C-b> %` လက်ရှိ pane ကို ဒေါင်လိုက် ပိုင်းခြားမည်
+ `<C-b> <direction>` သတ်မှတ်ထားသော _direction_ အတိုင်း pane သို့ ရွှေ့မည်။ ဤနေရာတွင် Direction ဆိုသည်မှာ မျှားကီးများကို ဆိုလိုသည်
+ `<C-b> z` လက်ရှိ pane အတွက် zoom ကို ဖွင့်/ပိတ် ပြုလုပ်မည်
+ `<C-b> <a href="https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/" class="ext-link">` Scrollback ကို စတင်မည်။ ထို့နောက် ရွေးချယ်မှု စတင်ရန် `<space>` ကို နှိပ်နိုင်ပြီး အဆိုပါ ရွေးချယ်မှုကို ကူးယူရန် `<enter>` ကို နှိပ်နိုင်သည်
+ `<C-b> <space>` Pane စီစဉ်ထားရှိမှု ပုံစံများကို အလှည့်ကျ ပြောင်းလဲမည်
```

> tmux အကြောင်း ပိုမို လေ့လာရန် [ဤ](https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) ရိုးရှင်း လွယ်ကူသော သင်ခန်းစာနှင့် [ဤ](https://linuxcommand.org/lc3_adv_termmux.php) ပိုမို အသေးစိတ်ကျသော ရှင်းလင်းချက်ကို ဖတ်ရှုရန် စဉ်းစားပါ။

သင့် toolkit တွင် tmux နှင့် SSH တို့ ပါဝင်လာပြီဖြစ်ရာ မည်သည့် စက်တွင်မဆို သင့် ပတ်ဝန်းကျင်ကို ကိုယ်ပိုင်အိမ်ကဲ့သို့ ခံစားရစေချင်ပါလိမ့်မည်။ ထိုနေရာတွင် shell စိတ်ကြိုက်ပြင်ဆင်ခြင်း (shell customization) ရောက်ရှိလာပါသည်။

Shell ကို စိတ်ကြိုက်ပြင်ဆင်ခြင်း

Command line ပရိုဂရမ် အမြောက်အများကို *dotfiles* ဟု သိကြသော ရိုးရိုးစာသား (plain-text) file များကို သုံး၍ ချိန်ညှိပြင်ဆင်ကြပါသည် (အဘယ်ကြောင့်ဆိုသော် file အမည်များသည် `.` ဖြင့် စတင်လေ့ရှိကြပြီး ဥပမာ `~/.vimrc` ဖြစ်သဖြင့် ၎င်းတို့ကို မူလအားဖြင့် `ls` directory စာရင်းထုတ်ရာတွင် ဝှက်ထားသောကြောင့် ဖြစ်သည်)။

> Dotfiles တွေဟာ အခြားသော shell သဘောတူညီချက် (convention) တစ်ခု ဖြစ်ပါတယ်။ အရှေ့မှ အစက်သည် စာရင်းထုတ်ရာတွင် ၎င်းတို့ကို "ဝှက်ထားရန်" ဖြစ်ပါတယ် (ဟုတ်ပါတယ်၊ အခြား convention တစ်ခုပါပဲ)။

Shell တွေဟာ ထိုကဲ့သို့သော file များဖြင့် ချိန်ညှိပြင်ဆင်ရသည့် ပရိုဂရမ်များ၏ ဥပမာတစ်ခု ဖြစ်ကြပါသည်။ စတင်ပွင့်ချိန်တွင် သင့် shell သည် ၎င်း၏ configuration များကို Load လုပ်ရန် file အများအပြားကို ဖတ်ရှုမည် ဖြစ်ပါသည်။ Shell နှင့် သင်သည် login နှင့်/သို့မဟုတ် interactive session တစ်ခုကို စတင်နေခြင်း ရှိမရှိ အပေါ် မူတည်၍ လုပ်ငန်းစဉ်တစ်ခုလုံးသည် တော်တော်လေး ရှုပ်ထွေးနိုင်ပါသည်။ [ဤနေရာတွင်](https://web.archive.org/web/20260329133158/https://blog.flowblok.id.au/2013-02/shell-startup-scripts.html) ဤ အကြောင်းအရာအတွက် အလွန်ကောင်းမွန်သော အရင်းအမြစ်တစ်ခု ရှိပါသည်။

`bash` အတွက် သင့် `.bashrc` သို့မဟုတ် `.bash_profile` ကို ပြင်ဆင်ခြင်းသည် စနစ်အများစုတွင် အလုပ်လုပ်ပါလိမ့်မည်။ Dotfiles များမှတစ်ဆင့် ချိန်ညှိပြင်ဆင်နိုင်သော အခြားသော tools များ၏ ဥပမာ အချို့မှာ -

- `bash` - `~/.bashrc` , `~/.bash_profile`
- `git` - `~/.gitconfig`
- `vim` - `~/.vimrc` နှင့် `~/.vim` folder
- `ssh` - `~/.ssh/config`
- `tmux` - `~/.tmux.conf`

အသုံးများသော configuration ပြောင်းလဲမှုတစ်ခုမှာ shell က ပရိုဂရမ်များကို ရှာဖွေနိုင်ရန် တည်နေရာအသစ်များ ပေါင်းစပ်ထည့်သွင်းပေးခြင်း ဖြစ်ပါသည်။ Software များကို install လုပ်သည့်အခါ ဤ pattern ကို တွေ့ကြုံရပါလိမ့်မည် -

```
export PATH="$PATH:path/to/append"
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် shell အား $PATH variable ၏ တန်ဖိုးကို ၎င်း၏ လက်ရှိတန်ဖိုးတွင် တည်နေရာအသစ် တစ်ခုပေါင်းစပ်၍ သတ်မှတ်ခိုင်းနေခြင်းဖြစ်ပြီး၊ child process အားလုံးသည် PATH အတွက် ဤတန်ဖိုးအသစ်ကို လက်ခံရရှိသွားမည် ဖြစ်ပါသည်။ ဒါဟာ child process များကို `path/to/append` အောက်တွင် တည်ရှိသော ပရိုဂရမ်များကို ရှာဖွေနိုင်စေမည် ဖြစ်ပါသည်။

သင့် shell ကို စိတ်ကြိုက်ပြင်ဆင်ခြင်းသည် command-line tools အသစ်များကို install လုပ်ခြင်းဟု မကြာခဏ အဓိပ္ပာယ်ရပါသည်။ Package manager များသည် ဤအရာကို လွယ်ကူစေပါသည်။ ၎င်းတို့သည် software များကို ဒေါင်းလုဒ်ဆွဲခြင်း၊ install လုပ်ခြင်းနှင့် update လုပ်ခြင်းတို့ကို စီမံဆောင်ရွက်ပေးကြပါသည်။ မတူညီသော operating system များတွင် မတူညီသော package manager များ ရှိကြပါသည် - macOS သည် **Homebrew** (https://brew.sh/) ကို သုံးသည်၊ Ubuntu/Debian တို့သည် `apt` ကို သုံးသည်၊ Fedora သည် `dnf` ကို သုံးပြီး Arch က `pacman` ကို သုံးပါသည်။ Shipping code သင်ခန်းစာတွင် package manager များကို အသေးစိတ် ထပ်မံ ဖော်ပြသွားပါမည်။

macOS တွင် Homebrew ကို အသုံးပြု၍ အသုံးဝင်သော tools နှစ်ခုကို မည်သို့ install လုပ်ရမည်ကို ဖော်ပြထားပါသည် -

```
# ripgrep: ပိုမိုကောင်းမွန်သော မူလပြင်ဆင်ချက်များ ပါဝင်ပြီး ပိုမိုမြန်ဆန်သော grep ဖြစ်သည်
brew install ripgrep

# fd: ပိုမိုမြန်ဆန်ပြီး သုံးစွဲသူအတွက် အဆင်ပြေသော find ဖြစ်သည်
brew install fd
```

ဤအရာများကို install လုပ်ပြီးပါက `grep` အစား `rg` ကိုလည်းကောင်း၊ `find` အစား `fd` ကို လည်းကောင်း အသုံးပြုနိုင်ပါပြီ။

> **`curl | bash` နှင့် ပတ်သက်၍ သတိပေးချက်** - `curl -fsSL https://example.com/install.sh | bash` ကဲ့သို့သော installation လွယ်ကြားချက်များကို မကြာခဏ တွေ့ရပါလိမ့်မည်။ ဤ pattern သည် script တစ်ခုကို ဒေါင်းလုဒ်လုပ်ပြီး ချက်ချင်း Run လိုက်ခြင်း ဖြစ်ရာ အဆင်ပြေသော်လည်း စွန့်စားမှု ရှိပါသည်။ အကြောင်းမှာ သင် စစ်ဆေးမထားသော စိုက်ထုတ်ထားသည့် code ကို Run နေခြင်းကြောင့် ဖြစ်သည်။ ပိုမို လုံခြုံသော နည်းလမ်းမှာ ပထမဆုံး ဒေါင်းလုဒ်လုပ်ရန်၊ ပြန်လည် သုံးသပ်ရန်၊ ထို့နောက်မှ Run ရန် ဖြစ်ပါသည် -
>
> ```
> curl -fsSL https://example.com/install.sh -o install.sh
> less install.sh  # script ကို ပြန်လည်သုံးသပ်ပါ
> bash install.sh
> ```
>
> အချို့သော installer များသည် အနည်းငယ် ပိုမိုလုံခြုံသော ပုံစံကွဲကို အသုံးပြုကြသည် - `/bin/bash -c "$(curl -fsSL https://url)"` ၎င်းသည် အနည်းဆုံးတော့ သင့်လက်ရှိ shell အစား bash က script ကို parse လုပ်ပေးကြောင်း သေချာစေပါသည်။

Install လုပ်မထားသော command တစ်ခုကို Run ရန် ကြိုးစားသောအခါ သင့် shell က `command not found` ဟု ဖော်ပြပါလိမ့်မည်။ ဝဘ်ဆိုက် **command-not-found.com** (https://command-not-found.com) သည် မတူညီသော package manager များနှင့် distribution များအလိုက် မည်သို့ install လုပ်ရမည်ဆိုသည်ကို ရှာဖွေနိုင်သည့် အသုံးဝင်သော အရင်းအမြစ်တစ်ခု ဖြစ်ပါသည်။

အခြား အသုံးဝင်သော tool တစ်ခုမှာ **`tldr`** (https://tldr.sh/) ဖြစ်ပြီး ၎င်းသည် ရိုးရှင်း၍ ဥပမာကို အဓိကထားသော man page များကို ပံ့ပိုးပေးပါသည်။ ရှည်လျားသော Documentation များကို ဖတ်ရှုမည့်အစား အသုံးများသော အသုံးပြုမှု pattern များကို လျှောက်ကြည့်နိုင်ပါသည် -

```
$ tldr fd
  An alternative to find.
  Aims to be faster and easier to use than find.

  Recursively find files matching a pattern in the current directory:
      fd "pattern"

  Find files that begin with "foo":
      fd "^foo"

  Find files with a specific extension:
      fd --extension txt
```

အချို့အခြေအနေများတွင် ပရိုဂရမ်အသစ် တစ်ခုလုံး မလိုအပ်ဘဲ သီးသန့် flag များပါသည့် လက်ရှိ command ၏ အတိုကောက် (shortcut) မျှသာ လိုအပ်တတ်ပါသည်။ ထိုနေရာတွင် alias များ ရောက်ရှိလာပါသည်။

Shell built-in ဖြစ်သော `alias` ကို အသုံးပြု၍ ကျွန်ုပ်တို့ကိုယ်ပိုင် command alias များကိုလည်း ဖန်တီးနိုင်ပါသည်။ Shell alias ဆိုသည်မှာ ၎င်း expression ကို မစစ်ဆေးမီ သင့် shell က အလိုအလျောက် အစားထိုးပေးမည့် အခြား command တစ်ခုအတွက် အတိုကောက် ပုံစံဖြစ်ပါသည်။ ဥပမာအားဖြင့် bash ရှိ alias တစ်ခုတွင် အောက်ပါ တည်ဆောက်ပုံ ပါရှိသည် -

```
alias alias_name="command_to_alias arg1 arg2"
```

> ညီမျှခြင်းသင်္ကေတ `=` ၏ ဘေးပတ်လည်တွင် space (ကွက်လပ်) မပါရှိကြောင်း သတိပြုပါ။ အကြောင်းမှာ **`alias`** (https://www.man7.org/linux/man-pages/man1/alias.1p.html) သည် argument တစ်ခု တည်းသာ ယူသော shell command တစ်ခု ဖြစ်သောကြောင့် ဖြစ်သည်။

Alias တွေမှာ အဆင်ပြေစေမည့် feature များစွာ ရှိကြပါတယ် -

```
# ဘုံသုံး flag များအတွက် အတိုကောက် ဖန်တီးပါ
alias ll="ls -lh"

# အသုံးများသော command များအတွက် စာရိုက်ရမှု အများအပြားကို သက်သာစေပါ
alias gs="git status"
alias gc="git commit"

# စာလုံးမှားယွင်း ရိုက်မိခြင်းမှ ကယ်တင်ပေးပါ
alias sl=ls

# ပိုမိုကောင်းမွန်သော မူလပြင်ဆင်ချက်များ ရရှိရန် လက်ရှိ command များကို ထပ်ရေးပါ
alias mv="mv -i"           # -i သည် အသစ်မထပ်ရေးမီ မေးမြန်းမည်ဖြစ်သည်
alias mkdir="mkdir -p"     # -p သည် လိုအပ်ပါက မိဘ dir များကို ဖန်တီးပေးမည်ဖြစ်သည်
alias df="df -h"           # -h သည် လူနားလည်လွယ်သော ပုံစံဖြင့် ရိုက်နှိပ်ပေးမည်ဖြစ်သည်

# Alias များကို ပေါင်းစပ်ဖွဲ့စည်းနိုင်သည်
alias la="ls -A"
alias lla="la -l"

# Alias ကို ပစ်ပယ်ထားရန် ရှေ့တွင် \ တပ်၍ Run ပါ
\ls
# သို့မဟုတ် unalias ဖြင့် alias ကို လုံးဝ ပိတ်ထားပါ
unalias la

# Alias သတ်မှတ်ချက်ကို ရယူရန် alias ဖြင့် ခေါ်ယူရုံမျှသာ ပြုလုပ်ပါ
alias ll
# ll='ls -lh' ဟု ရိုက်နှိပ်မည်ဖြစ်သည်
```

Alias များတွင် ကန့်သတ်ချက်များ ရှိသည် - ၎င်းတို့သည် command ၏ အလယ်တွင် argument များကို ယူ၍ မရပါ။ ပိုမို ရှုပ်ထွေးသော ပြုမူဆောင်ရွက်ချက်များအတွက် ၎င်းအစား shell function များကို သုံးသင့်ပါသည်။

Shell အများစုသည် ရာဇဝင် (history) ကို နောက်ပြန် ရှာဖွေရန်အတွက် `Ctrl-R` ကို ပံ့ပိုးပေးထားပါသည်။ `Ctrl-R` ကို နှိပ်ပြီး ယခင် command များကို ရှာဖွေရန် စာရိုက်ပါ။ အစောပိုင်းက fuzzy finder အဖြစ် `fzf` ကို မိတ်ဆက်ခဲ့ပါသည်၊ fzf ၏ shell integration ဖြင့် ချိန်ညှိထားပါက `Ctrl-R` သည် မူလပုံစံထက် မက ပိုမို စွမ်းအားထက်မြက်သည့် သင့် history တစ်ခုလုံးအပေါ် အပြန်အလှန်လုပ်ဆောင်နိုင်သော fuzzy search တစ်ခု ဖြစ်လာပါသည်။

သင့် dotfile များကို မည်သို့ စုစည်းသင့်သနည်း။ ၎င်းတို့သည် ကိုယ်ပိုင် folder တွင် ရှိသင့်ပြီး၊ version control အောက်တွင် ရှိကာ script ကို သုံး၍ နေရာတကျ **symlink** ပြုလုပ်ထားသင့်ပါသည်။ ဒါဟာ အောက်ပါ အကျိုးကျေးဇူးများ ရှိစေပါသည် -

- **တပ်ဆင်ရ လွယ်ကူခြင်း** - အကယ်၍ စက်အသစ်တစ်ခုသို့ ဝင်ရောက်ပါက သင့် စိတ်ကြိုက်ပြင်ဆင်ချက်များကို ထည့်သွင်းရန် တစ်မိနစ်မျှသာ ကြာမြင့်မည် ဖြစ်ပါသည်။
- **သယ်ဆောင်လွယ်ခြင်း (Portability)** - သင့် tools များသည် နေရာတိုင်းတွင် တူညီသော ပုံစံဖြင့် အလုပ်လုပ်ပါလိမ့်မည်။
- **ဟန်ချက်ညီ အပြိုင်ဖြစ်ခြင်း (Synchronization)** - မည်သည့်နေရာတွင်မဆို သင့် dotfile များကို update လုပ်နိုင်ပြီး ၎င်းတို့အားလုံးကို အပြိုင် ဟန်ချက်ညီစေနိုင်ပါသည်။
- **ပြောင်းလဲမှု မှတ်တမ်းတင်နိုင်ခြင်း** - သင့် ပရိုဂရမ်မင်း သက်တမ်းတစ်ခုလုံးတွင် သင့် dotfile များကို ထိန်းသိမ်းသွားဖွယ် ရှိရာ ကြာရှည်သုံးမည့် project များအတွက် version history ရှိထားခြင်းမှာ ကောင်းမွန်ပါသည်။

သင့် dotfile တွေထဲမှာ မည်သည့်အရာတွေ ထည့်ရမလဲ။ အွန်လိုင်း Documentation များ သို့မဟုတ် **man page** (https://en.wikipedia.org/wiki/Man_page) များကို ဖတ်ရှုခြင်းဖြင့် သင့် tool ၏ setting များကို လေ့လာနိုင်ပါသည်။ အခြား နည်းလမ်းကောင်းတစ်ခုမှာ စာရေးသူများက ၎င်းတို့ နှစ်သက်သော စိတ်ကြိုက်ပြင်ဆင်မှုများအကြောင်း ပြောပြထားသည့် သီးသန့် ပရိုဂရမ်များဆိုင်ရာ blog post များကို အင်တာနက်တွင် ရှာဖွေခြင်းဖြစ်ပါသည်။ စိတ်ကြိုက်ပြင်ဆင်မှုများအကြောင်း လေ့လာရန် အခြားနည်းလမ်းတစ်ခုမှာ အခြားသူများ၏ dotfile များကို လျှောက်ကြည့်ခြင်း ဖြစ်ပါသည် - GitHub တွင် **dotfiles repository** (https://github.com/search?o=desc&q=dotfiles&s=stars&type=Repositories) မြောက်မြားစွာကို ရှာဖွေတွေ့ရှိနိုင်ပါသည် — လူကြိုက်အများဆုံးတစ်ခုကို ဤနေရာတွင် (https://github.com/mathiasbynens/dotfiles) ကြည့်ရှုနိုင်ပါသည် (သို့သော် configuration များကို မျက်စိမှိတ် မကူးယူရန် အကြံပြုပါသည်)။ ဤနေရာတွင် (https://dotfiles.github.io/) ဤအကြောင်းအရာနှင့် ပတ်သက်သည့် အခြား ကောင်းမွန်သော အရင်းအမြစ်တစ်ခု ရှိပါသည်။

သင်တန်းဆရာများ အားလုံး၏ dotfile များကို GitHub တွင် အများပြည်သူ ကြည့်ရှုနိုင်ရန် ဖွင့်လှစ်ပေးထားပါသည် - **Anish** (https://github.com/anishathalye/dotfiles), **Jon** (https://github.com/jonhoo/configs), **Jose** (https://github.com/jjgo/dotfiles)။

Framework များနှင့် plugin များ သည်လည်း သင့် shell ကို ပိုမိုကောင်းမွန်စေနိုင်ပါသည်။ အသုံးများသော အထွေထွေ framework အချို့မှာ **prezto** (https://github.com/sorin-ionescu/prezto) သို့မဟုတ် **oh-my-zsh** (https://ohmyz.sh/) တို့ဖြစ်ကြပြီး၊ သီးသန့် feature များကို အဓိကထားသည့် သေးငယ်သော plugin များမှာ -

- **zsh-syntax-highlighting** (https://github.com/zsh-users/zsh-syntax-highlighting) - စာရိုက်နေစဉ် တရားဝင်/မတရားဝင် command များကို အရောင်ဆိုးပေးသည်
- **zsh-autosuggestions** (https://github.com/zsh-users/zsh-autosuggestions) - စာရိုက်နေစဉ် history မှ command များကို အကြံပြုပေးသည်
- **zsh-completions** (https://github.com/zsh-users/zsh-completions) - အပိုထပ်ဆောင်း ဖြည့်စွက်ချက် (completion) သတ်မှတ်ချက်များ

- **zsh-history-substring-search** (https://github.com/zsh-users/zsh-history-substring-search) - fish ကဲ့သို့သော history ရှာဖွေမှု
- **powerlevel10k** (https://github.com/romkatv/powerlevel10k) - မြန်ဆန်ပြီး စိတ်ကြိုက်ပြင်နိုင်သော prompt theme

fish (https://fishshell.com/) ကဲ့သို့သော shell တွင် ဤ feature အများအပြားကို မူလအားဖြင့် ထည့်သွင်းပေးထားပါသည်။

> ဤ feature များကို ရရှိရန် oh-my-zsh ကဲ့သို့သော ကြီးမားသည့် framework တစ်ခု မလိုအပ်ပါ။ တစ်သီးပုဂ္ဂလ plugin များကို install လုပ်ခြင်းသည် မကြာခဏ ပိုမိုမြန်ဆန်ပြီး ပိုမို ထိန်းချုပ်ခွင့် ပေးပါသည်။ ကြီးမားသော framework များသည် shell စတင်ချိန်ကို သိသာစွာ နှေးကွေးစေနိုင်သဖြင့် သင် အမှန်တကယ် သုံးစွဲသည့်အရာများကိုသာ install လုပ်ရန် စဉ်းစားပါ။

Shell အတွင်း AI အသုံးပြုခြင်း

Shell တွင် AI tools များကို ထည့်သွင်းအသုံးပြုရန် နည်းလမ်းများစွာ ရှိပါသည်။ အောက်တွင် မတူညီသော ပေါင်းစပ်မှု အဆင့်များအလိုက် ဥပမာအချို့ကို ဖော်ပြထားပါသည် -

Command ထုတ်လုပ်ပေးခြင်း - `simonw/llm` (https://github.com/simonw/llm) ကဲ့သို့သော tools များသည် သဘာဝဘာသာစကား ဖော်ပြချက်များမှတစ်ဆင့် shell command များကို ထုတ်လုပ်ပေးနိုင်ရန် ကူညီပေးနိုင်ပါသည်။

```
$ llm cmd "find all python files modified in the last week"
find . -name "*.py" -mtime -7
```

Pipeline တွင် ပေါင်းစပ်ခြင်း - LLM များကို ဒေတာများကို ပရိုဆက်လုပ်ရန်နှင့် ပြောင်းလဲရန် shell pipeline များအတွင်း ပေါင်းစပ်နိုင်ပါသည်။ ရိုးရိုး regex သုံးလျှင် ခက်ခဲမည့် ပုံစံမမှန်သော ဖော်မတ်များမှ အချက်အလက်များကို ထုတ်ယူရန် လိုအပ်သည့်အခါ ၎င်းတို့သည် အထူး အသုံးဝင်ပါသည် -

```
$ cat users.txt
Contact: john.doe@example.com
User 'alice_smith' logged in at 3pm
Posted by: @bob_jones on Twitter
Author: Jane Doe (jdoe)
Message from mike_wilson yesterday
Submitted by user: sarah.connor
$ INSTRUCTIONS="Extract just the username from each line, one per line, nothing
else"
$ llm "$INSTRUCTIONS" < users.txt
john.doe
alice_smith
bob_jones
jdoe
mike_wilson
sarah.connor
```

Variable တွင် space (ကွက်လပ်) များ ပါဝင်နေသောကြောင့် `"$INSTRUCTIONS"` (quote ဖြင့်) ကို သုံးထားပုံနှင့် file ၏ အကြောင်းအရာကို stdin သို့ လမ်းကြောင်းပြောင်းရန် `< users.txt` ကို သုံးထားပုံကို သတိပြုပါ။

AI Shell များ - [Claude Code](https://docs.anthropic.com/en/docs/claude-code) (https://docs.anthropic.com/en/docs/claude-code) ကဲ့သို့သော tools များသည် အင်္ဂလိပ်စာ command များကို လက်ခံပြီး ၎င်းတို့ကို shell လုပ်ဆောင်ချက်များ၊ file ပြင်ဆင်မှုများနှင့် ပိုမို ရှုပ်ထွေးသော အဆင့်များစွာပါဝင်သည့် အလုပ်များအဖြစ် ဘာသာပြန်ပေးသည့် meta-shell အဖြစ် လုပ်ဆောင်ပေးပါသည်။

Terminal Emulator များ

သင့် shell ကို စိတ်ကြိုက်ပြင်ဆင်ခြင်းနှင့်အတူ သင့်အနေဖြင့် **terminal emulator** ရွေးချယ်မှုနှင့် ၎င်း၏ setting များကို စူးစမ်းရန် အချိန်အနည်းငယ် ပေးသင့်ပါသည်။ Terminal emulator ဆိုသည်မှာ သင့် shell Run နေသည့် စာသားအခြေပြု interface ကို ပံ့ပိုးပေးသော GUI ပရိုဂရမ်တစ်ခု ဖြစ်ပါသည်။ ထွက်ရှိထားသော terminal emulator များစွာ ရှိကြပါသည်။

သင့် terminal တွင် နာရီပေါင်း ရာနှင့်ချီ၍ သို့မဟုတ် ထောင်နှင့်ချီ၍ အချိန်ကုန်လွန်ရဖွယ် ရှိသဖြင့် ၎င်း၏ setting များကို ဝင်ရောက်ကြည့်ရှုရန် ထိုက်တန်လှပါသည်။ သင့် terminal တွင် ပြုပြင်ပြောင်းလဲလိုနိုင်သည့် အချက်အချို့တွင် အောက်ပါတို့ ပါဝင်သည် -

- Font ရွေးချယ်မှု
- Color Scheme
- Keyboard shortcut များ
- Tab/Pane ထောက်ပံ့မှု
- Scrollback configuration
- စွမ်းဆောင်ရည် (**Alacritty** (https://github.com/alacritty/alacritty) သို့မဟုတ် **Ghostty** (https://ghostty.org/) ကဲ့သို့သော အချို့သော သစ်လွင်သည့် terminal များသည် GPU acceleration ကို ပံ့ပိုးပေးကြသည်)။

လေ့ကျင့်ခန်းများ (Exercises)

Arguments များနှင့် Globs တွဲဖက်သုံးခြင်း

1. သင့်အနေဖြင့် `cmd --flag -- --notaflag` ကဲ့သို့သော command များကို တွေ့ရနိုင်ပါသည်။ `--` သည် ပရိုဂရမ်အား flag များကို parse လုပ်ခြင်း ရပ်တန့်ရန် ပြောဆိုသော အထူး argument ဖြစ်ပါသည်။ `--` ၏ အနောက်ရှိ အရာအားလုံးကို positional argument အဖြစ် သတ်မှတ်ပါသည်။ ဒါဟာ အဘယ်ကြောင့် အသုံးဝင်နိုင်သနည်း။ `touch -- -myfile` ကို Run ကြည့်ပြီး `--` မပါဘဲ ၎င်းကို ပြန်လည် ဖျက်ပစ်ရန် ကြိုးစားကြည့်ပါ။

2. **`man ls`** (https://www.man7.org/linux/man-pages/man1/ls.1.html) ကို ဖတ်ရှုပြီး file များကို အောက်ပါ ပုံစံအတိုင်း စာရင်းထုတ်ပေးသည့် `ls` command တစ်ခု ရေးသားပါ -

```
- ဝှက်ထားသော file များ ပါဝင်သော file အားလုံး ပါဝင်ရမည်
- အရွယ်အစားများကို လူနားလည်လွယ်သော ပုံစံဖြင့် စာရင်းဖော်ပြရမည် (ဥပမာ 454279954 အစား 454M)
- File များကို လတ်တဆတ်ဆုံး အစီအစဉ်အတိုင်း စီစဉ်ထားရမည်
- Output ကို အရောင်ဆိုးထားရမည်

နမူနာ output သည် အောက်ပါအတိုင်း ဖြစ်ပါလိမ့်မည် -
```

```
```
-rw-r--r--   1 user group 1.1M Jan 14 09:53 baz
drwxr-xr-x   5 user group 160 Jan 14 09:53 .
-rw-r--r--   1 user group 514 Jan 14 06:42 bar
-rw-r--r--   1 user group 106M Jan 13 12:12 foo
drwx------+ 47 user group 1.5K Jan 12 18:08 ..
```
```

1. Process substitution `<(command)` သည် command တစ်ခု၏ output ကို file တစ်ခုကဲ့သို့ အသုံးပြုခွင့်ပေးပါသည်။ `printenv` နှင့် `export` တို့၏ output များကို နှိုင်းယှဉ်ရန် process substitution နှင့် အတူ `diff` ကို အသုံးပြုပါ။ ၎င်းတို့ အဘယ်ကြောင့် မတူညီကြသနည်း။ (အကြံပြုချက် - `diff <(printenv | sort) <(export | sort)` ကို စမ်းကြည့်ပါ)။

Environment Variables များ

1. အောက်ပါအတိုင်း လုပ်ဆောင်ပေးသော bash function `marco` နှင့် `polo` တို့ကို ရေးသားပါ - သင့်အနေဖြင့် `marco` ကို Run သည့်အခါတိုင်း လက်ရှိ working directory ကို တစ်နည်းနည်းဖြင့် သိမ်းဆည်းထားရမည်၊ ထို့နောက် မည်သည့် directory သို့ ရောက်ရှိနေပါစေ `polo` ကို Run လိုက်ပါက `polo` က သင့်အား `marco` ကို Run ခဲ့သည့် directory သို့ `cd` ဖြင့် ပြန်လည် ပို့ဆောင်ပေးရမည်။ Debug ပြုလုပ်ရန် လွယ်ကူစေရေးအတွက် code ကို `marco.sh` file တစ်ခုတွင် ရေးသားနိုင်ပြီး `source marco.sh` ကို Run ခြင်းဖြင့် သင့် shell သို့ သတ်မှတ်ချက်များကို (ပြန်လည်) Load ပြုလုပ်နိုင်ပါသည်။

Return Codes များ

1. သင့်တွင် ခဲယဉ်းစွာ မအောင်မြင်တတ်သော command တစ်ခု ရှိသည် ဆိုပါစို့။ ၎င်းကို debug ပြုလုပ်ရန် output ကို ဖမ်းယူရန် လိုအပ်သော်လည်း မအောင်မြင်သော Run ကို ရရှိရန် အချိန်ကုန်နိုင်ပါသည်။ အောက်ပါ script ကို မအောင်မြင်မချင်း Run ပြီး ၎င်း၏ standard output နှင့် error stream များကို file များသို့ ဖမ်းယူကာ အဆုံးတွင် အရာအားလုံးကို ရိုက်နှိပ်ပေးသည့် bash script တစ်ခု ရေးသားပါ။ Script မအောင်မြင်မီ အရေအတွက် မည်မျှ Run ခဲ့ရသည်ကိုလည်း သတင်းပို့နိုင်ပါက Bonus အမှတ် ရရှိပါမည်။

```
#!/usr/bin/env bash

n=$(( RANDOM % 100 ))

if [[ $n -eq 42 ]]; then
   echo "Something went wrong"
   >&2 echo "The error was using magic numbers"
   exit 1
fi

echo "Everything went according to plan"
```

Signal များနှင့် Job Control

1. Terminal ထဲတွင် `sleep 10000` job တစ်ခုကို စတင်ပါ၊ `Ctrl-Z` ဖြင့် background သို့ ပို့ပြီး `bg` ဖြင့် ၎င်း၏ လုပ်ဆောင်မှုကို ဆက်လက် ပြုလုပ်ပါ။ ယခုအခါ ၎င်း၏ pid ကို ရှာရန် **`pgrep`** (https://www.man7.org/linux/man-pages/man1/pgrep.1.html) ကို အသုံးပြုပြီး pid ကိုယ်တိုင် ရိုက်ထည့်ရန် မလိုဘဲ ၎င်းအား kill ပြုလုပ်ရန် **`pkill`** (https://man7.org/linux/man-pages/man1/pgrep.1.html) ကို အသုံးပြုပါ။ (အကြံပြုချက် - `-lf` flag များကို သုံးပါ)။

2. အခြား process တစ်ခု မပြီးဆုံးမီ process တစ်ခုကို မစတင်ချင်ဟု ဆိုပါစို့။ မည်သို့ ပြုလုပ်မည်နည်း။ ဤလေ့ကျင့်ခန်းတွင် ကျွန်ုပ်တို့၏ ကန့်သတ်ချက် process သည် အမြဲတမ်း `sleep 60 &` ဖြစ်ပါမည်။ ဤအရာကို ရရှိရန် နည်းလမ်းတစ်ခုမှာ `wait` (https://www.man7.org/linux/man-pages/man1/wait.1p.html) command ကို အသုံးပြုခြင်း ဖြစ်ပါသည်။ Sleep command ကို စတင်ပြီး background process မပြီးဆုံးမီအထိ `ls` တစ်ခုက စောင့်ဆိုင်းအောင် ပြုလုပ်ကြည့်ပါ။

> သို့သော် ဤနည်းဗျူဟာသည် မတူညီသော bash session တစ်ခုတွင် စတင်ပါက ပျက်ကွက်ပါလိမ့်မည်၊ အကြောင်းမှာ `wait` သည် child process များအတွက်သာ အလုပ်လုပ်သောကြောင့် ဖြစ်သည်။ မှတ်စုများတွင် ကျွန်ုပ်တို့ မဆွေးနွေးခဲ့သည့် feature တစ်ခုမှာ `kill` command ၏ exit status သည် အောင်မြင်ပါက သုညဖြစ်ပြီး သို့မဟုတ်ပါက သုညမဟုတ်ဘဲ ဖြစ်နေခြင်းပင် ဖြစ်သည်။ `kill -0` သည် signal တစ်ခုကို မပို့သော်လည်း process မရှိပါက သုညမဟုတ်သော exit status ကို ပေးပါလိမ့်မည်။ Pid တစ်ခုကို ယူပြီး ပေးထားသော process မပြီးဆုံးမီအထိ စောင့်ဆိုင်းပေးသည့် `pidwait` ဟု ခေါ်သော bash function တစ်ခု ရေးသားပါ။ CPU ကို မလိုအပ်ဘဲ ဖြုန်းတီးခြင်းမှ ရှောင်ရှားရန် `sleep` ကို အသုံးပြုသင့်ပါသည်။

File များနှင့် Permission များ

1. (အဆင့်မြင့်) Directory တစ်ခုအတွင်း လတ်တဆတ်ဆုံး ပြင်ဆင်ထားသော file ကို recursive နည်းဖြင့် ရှာဖွေရန် command သို့မဟုတ် script တစ်ခု ရေးသားပါ။ ပိုမို အထွေထွေကျစွာဖြင့် file အားလုံးကို လတ်တဆတ်ဆုံး အစီအစဉ်အတိုင်း စာရင်းထုတ်နိုင်ပါသလား။

Terminal Multiplexers

1. ဤ `tmux` [သင်ခန်းစာ](https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) (https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) ကို လိုက်နာပါ၊ ထို့နောက် [ဤအဆင့်များ](https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/) (https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/) ကို လိုက်နာ၍ အခြေခံ စိတ်ကြိုက်ပြင်ဆင်မှုများ ပြုလုပ်နည်းကို လေ့လာပါ။

Alias များနှင့် Dotfiles

1. စာလုံးမှား ရိုက်မိသည့်အခါ `cd` သို့ ရောက်ရှိသွားမည့် alias `dc` တစ်ခု ဖန်တီးပါ။

2. သင့်အသုံးအများဆုံး ထိပ်တန်း command ၁၀ ခုကို ရရှိရန် `history | awk '{$1="";print substr($0,2)}' | sort | uniq -c | sort -n | tail -n 10` ကို Run ပြီး ၎င်းတို့အတွက် ပိုမို တိုတောင်းသော alias များ ရေးသားရန် စဉ်းစားပါ။ မှတ်ချက် - ဤအရာသည် Bash အတွက် အလုပ်လုပ်ပါသည်၊ အကယ်၍ ZSH ကို သုံးနေပါက `history` အစား `history 1` ကို သုံးပါ။

3. သင့် dotfile များအတွက် folder တစ်ခု ဖန်တီးပြီး version control ထူထောင်ပါ။

4. စိတ်ကြိုက်ပြင်ဆင်မှု အချို့ ပါရှိသော အနည်းဆုံး ပရိဂရမ်တစ်ခု (ဥပမာ သင့် shell) အတွက် configuration တစ်ခု ထည့်သွင်းပါ (စတင်ရန်အတွက် `$PS1` သတ်မှတ်ခြင်းဖြင့် သင့် shell prompt ကို စိတ်ကြိုက်ပြင်ခြင်းကဲ့သို့ ရိုးရှင်းသော အရာတစ်ခု ဖြစ်နိုင်ပါသည်)။

5. စက်အသစ်တွင် သင့် dotfile များကို လျှင်မြန်စွာ (နှင့် ကိုယ်တိုင် အားထုတ်စရာမလိုဘဲ) install လုပ်မည့် နည်းလမ်းတစ်ခု ထူထောင်ပါ။ ဒါဟာ file တစ်ခုစီအတွက် `ln -s` ကို ခေါ်ယူသည့် ရိုးရှင်းသော shell script တစ်ခု ဖြစ်နိုင်သည်၊ သို့မဟုတ် [သီးသန့် utility](https://dotfiles.github.io/utilities/) တစ်ခုကို သုံးနိုင်ပါသည်။

6. သန့်ရှင်းသော virtual machine အသစ်တစ်ခုတွင် သင့် installation script ကို စမ်းသပ်ပါ။

7. သင့် လက်ရှိ tool configuration အားလုံးကို သင့် dotfiles repository ထဲ သို့ ပြောင်းရွှေ့ပါ။

8. သင့် dotfile များကို GitHub တွင် ထုတ်ဝေပါ။

Remote Machine များ (SSH)

ဤလေ့ကျင့်ခန်းများအတွက် Linux virtual machine တစ်ခု ထည့်သွင်းပါ (သို့မဟုတ် ရှိနှင့်ပြီးသား တစ်ခုကို သုံးပါ)။ အကယ်၍ သင့်အနေဖြင့် virtual machine များနှင့် မရင်းနှီးပါက တစ်ခု ထည့်သွင်းရန်အတွက် ဤသင်ခန်းစာကို ကြည့်ရှုပါ။

1. `~/.ssh/` သို့ သွားပြီး သင့်တွင် SSH key တွဲတစ်ခု ရှိမရှိ စစ်ဆေးပါ။ မရှိပါက `ssh-keygen -a 100 -t ed25519` ဖြင့် ထုတ်လုပ်ပါ။ Password အသုံးပြုရန်နှင့် `ssh-agent` ကို သုံးရန် အကြံပြုပါသည်၊ အချက်အလက် ထပ်မံကြည့်ရှုရန် ဤနေရာတွင် (https://www.ssh.com/ssh/agent) ကြည့်ပါ။

2. အောက်ပါအတိုင်း ပါဝင်အောင် `.ssh/config` ကို ပြင်ဆင်ပါ -

```
Host vm
    User username_goes_here
    HostName ip_goes_here
    IdentityFile ~/.ssh/id_ed25519
    LocalForward 9999 localhost:8888
```

3. Server ထံသို့ သင့် ssh key ကူးယူရန် `ssh-copy-id vm` ကို သုံးပါ။

4. သင့် VM တွင် `python -m http.server 8888` ကို Run ၍ webserver တစ်ခု စတင်ပါ။ သင့်စက်ရှိ `http://localhost:9999` သို့ ဝင်ရောက်၍ VM webserver ကို ကြည့်ရှုပါ။

5. `sudo vim /etc/ssh/sshd_config` ပြုလုပ်၍ သင့် SSH server config ကို ပြင်ဆင်ပြီး `PasswordAuthentication` တန်ဖိုးကို ပြင်ဆင်၍ password authentication ကို ပိတ်ထားပါ။ `PermitRootLogin` တန်ဖိုးကို ပြင်ဆင်၍ root login ကို ပိတ်ထားပါ။ `sudo service sshd restart` ဖြင့် `ssh` service ကို restart ပြုလုပ်ပါ။ SSH ပြန်ဝင်ကြည့်ပါ။

6. (စိန်ခေါ်မှု) VM တွင် **`mosh`** (https://mosh.org/) ကို install လုပ်ပြီး ချိတ်ဆက်မှု ထူထောင်ပါ။ ထို့နောက် server/VM ၏ network adapter ကို ဖြုတ်လိုက်ပါ။ Mosh က ၎င်းမှ မှန်ကန်စွာ ပြန်လည် သက်သာလာနိုင်ပါသလား။

7. (စိန်ခေါ်မှု) `ssh` တွင် `-N` နှင့် `-f` flag များ အဘယ်သို့ ပြုလုပ်သည်ကို ရှာဖွေပြီး background port forwarding ကို ရရှိရန် command တစ်ခုကို စိစစ်ဖော်ထုတ်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. `fzf`	https://github.com/junegunn/fzf	
2. short-circuiting	https://en.wikipedia.org/wiki/Short-circuit_evaluation	
3. `kill`	https://www.man7.org/linux/man-pages/man1/kill.1.html	
4. `fg`	https://www.man7.org/linux/man-pages/man1/fg.1p.html	
5. `bg`	https://man7.org/linux/man-pages/man1/bg.1p.html	
6. `jobs`	https://www.man7.org/linux/man-pages/man1/jobs.1p.html	
7. `pgrep`	https://www.man7.org/linux/man-pages/man1/pgrep.1.html	
8. `nohup`	https://www.man7.org/linux/man-pages/man1/nohup.1.html	
9. ဤနေရာတွင်	https://en.wikipedia.org/wiki/Signal_(IPC	
10. `man signal`	https://www.man7.org/linux/man-pages/man1/signal.7.html	
11. Mosh	https://mosh.org/	
12. `ssh-keygen`	https://www.man7.org/linux/man-pages/man1/ssh-keygen.1.html	
13. ဤနေရာတွင်	https://help.github.com/articles/connecting-to-github-with-ssh/	
14. `scp`	https://www.man7.org/linux/man-pages/man1/scp.1.html	
15. `rsync`	https://www.man7.org/linux/man-pages/man1/rsync.1.html	

Resource / Description	URL	Micro QR
16. `tmux`	https://www.man7.org/linux/man-pages/man1/tmux.1.html	
17. ` Scrollback ကို စတင်မည်။ ထို့နောက် ရွေးချယ်မှု စတင်ရန် `` ကို နှိပ်နှိုင်ပြီး အဆုံပါ ရွေးချယ်မှု ကို ကူးယူရန် `` ကို နှိပ်နှိုင်သည် + ``` Pane စီစဉ်ထားရှိမှု ပုံစံများကို အလှည့်ကျ ပြောင်းလဲမည် > tmux အကြောင်း ပိုမို လေ့လာရန် ဤ	https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/	
18. ဤ	https://linuxcommand.org/lc3_adv_termmux.php	
19. ဤနေရာတွင်	https://web.archive.org/web/20260329133158/https://blog.flowblok.id.au/2013-02/shell-startup-scripts.html	
20. Homebrew	https://brew.sh/	
21. command-not-found.com	https://command-not-found.com	
22. `tldr`	https://tldr.sh/	
23. `alias`	https://www.man7.org/linux/man-pages/man1/alias.1p.html	
24. man page	https://en.wikipedia.org/wiki/Man_page	
25. dotfiles repository	https://github.com/search?o=desc&q=dotfiles&s=stars&type=Repositories	
26. ဤနေရာတွင်	https://github.com/mathiasbynens/dotfiles	
27. ဤနေရာတွင်	https://dotfiles.github.io/	
28. Anish	https://github.com/anishathalye/dotfiles	
29. Jon	https://github.com/jonhoo/configs	

Resource / Description	URL	Micro QR
30. Jose	https://github.com/jjgo/dotfiles	
31. prezto	https://github.com/sorin-ionescu/prezto	
32. oh-my-zsh	https://ohmyz.sh/	
33. zsh-syntax-highlighting	https://github.com/zsh-users/zsh-syntax-highlighting	
34. zsh-autosuggestions	https://github.com/zsh-users/zsh-autosuggestions	
35. zsh-completions	https://github.com/zsh-users/zsh-completions	
36. zsh-history-substring-search	https://github.com/zsh-users/zsh-history-substring-search	
37. powerlevel10k	https://github.com/romkatv/powerlevel10k	
38. fish	https://fishshell.com/	
39. `simonw/llm`	https://github.com/simonw/llm	
40. Claude Code	https://docs.anthropic.com/en/docs/claude-code	
41. Alacritty	https://github.com/alacritty/alacritty	
42. Ghostty	https://ghostty.org/	
43. `man ls`	https://www.man7.org/linux/man-pages/man1/ls.1.html	
44. `pkill`	https://man7.org/linux/man-pages/man1/pgrep.1.html	
45. `wait`	https://www.man7.org/linux/man-pages/man1/wait.1p.html	

Resource / Description	URL	Micro QR
46. ဤအဆင့်များ	https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/	
47. သီးသန့် utility	https://dotfiles.github.io/utilities/	
48. ဤနေရာတွင်	https://www.ssh.com/ssh/agent	

```
\pagebreak

# သင်တန်းမိတ်ဆက် + Shell အသုံးပြုနည်းမိတ်ဆက်

> **အနှစ်ချုပ်** - ဤသင်တန်းကို ဖွင့်လှစ်သင်ကြားရခြင်း ရည်ရွယ်ချက်ကို လေ့လာပြီး Shell ကို စတင်အသုံးပြုပါ။


```{=html}
<div class="video-card">
  <div class="video-card-header">
    <span class="video-badge">▶ LECTURE VIDEO REFERENCE</span>
    <h3 class="video-title">သင်တန်းမိတ်ဆက် + Shell အသုံးပြုနည်းမိတ်ဆက်</h3>
  </div>
  <div class="video-card-body">
    <div class="video-preview-container">
      <img src="file:///home/runner/work/missing-semester-my.github.io/missing-
semester-my.github.io/ebook/assets/thumbnails/yt_MSgoeuMqUmU.jpg" alt="သင်တန်းမိ
တ်ဆက် + Shell အသုံးပြုနည်းမိတ်ဆက် Preview" class="video-thumbnail" />
    </div>
    <div class="video-info-container">
      <p class="video-desc">Scan the QR code below using your mobile camera or
click the link to watch this full lecture online:</p>
      <div class="video-qr-block">
        <img src="file:///home/runner/work/missing-semester-my.github.io/missin
g-semester-my.github.io/ebook/assets/qrcodes/yt_MSgoeuMqUmU.png" alt="QR Code f
or သင်တန်းမိတ်ဆက် + Shell အသုံးပြုနည်းမိတ်ဆက်" class="video-qr-code" />
        <div class="video-url-details">
          <span class="qr-label">Direct Online Link:</span>
          <a href="https://www.youtube.com/watch?v=MSgoeuMqUmU" class="video-ur
l">https://www.youtube.com/watch?v=MSgoeuMqUmU</a>
        </div>
      </div>
    </div>
  </div>
</div>
```
```

ကျွန်ုပ်တို့သည် မည်သူများနည်း။

ဤသင်တန်းကို **Anish** (https://anish.io/)၊ **Jon** (https://thesquareplanet.com/) နှင့် **Jose** (http://josejg.com/) တို့မှ ပူးတွဲသင်ကြားပေးထားခြင်းဖြစ်သည်။ ကျွန်ုပ်တို့အားလုံးသည် MIT ကျောင်းသားဟောင်းများဖြစ်ကြပြီး ကျောင်းသားဘဝတုန်းက ဤ MIT IAP သင်တန်းကို စတင်ခဲ့ကြခြင်းဖြစ်သည်။ ကျွန်ုပ်တို့ထံ ဆက်သွယ်လိုပါက **missing-semester@mit.edu** သို့ အီးမေးလ်ပေးပို့နိုင်ပါသည်။

ကျွန်ုပ်တို့သည် ဤသင်တန်းကို သင်ကြားရန်အတွက် ဉာဏ်ပူဇော်ခ ရယူခြင်းမရှိသလို၊ မည်သည့်နည်းလမ်းဖြင့်မျှ စီးပွားဖြစ် ပြုလုပ်ထားခြင်းမရှိပါ။ ကျွန်ုပ်တို့၏ သင်တန်းစာသင်ပစ္စည်းများ (https://missing.csail.mit.edu/) နှင့် သင်ခန်းစာ ဗီဒီယိုမှတ်တမ်းများ (https://www.youtube.com/@MissingSemester) အားလုံးကို အွန်လိုင်းတွင် အခမဲ့ လွတ်လပ်စွာ ကြည့်ရှုနိုင်ရန် စီစဉ်ပေးထားပါသည်။ သင်၏ပံ့ပိုးကူညီမှုကို ပြသလိုပါက အကောင်းဆုံးနည်းလမ်းမှာ ဤသင်တန်းအကြောင်းကို အခြားသူများထံ သတင်းမျှဝေပေးခြင်းပင် ဖြစ်ပါသည်။ အကယ်၍ သင်သည် ကုမ္ပဏီ၊ တက္ကသိုလ် သို့မဟုတ် အခြားအဖွဲ့အစည်းတစ်ခုဖြစ်ပြီး ဤသင်ရိုးညွှန်းတမ်းကို သင်တန်းသားအများအပြားအား သင်ကြားပို့ချပေးနေပါက ကျွန်ုပ်တို့သိရှိနိုင်ရန်အတွက် အတွေ့အကြုံ အစီရင်ခံစာများ/သုံးသပ်ချက်များကို အီးမေးလ်မှတစ်ဆင့် ပေးပို့ပေးပါရန် မေတ္တာရပ်ခံအပ်ပါသည်။ :)

သင်တန်းဖွင့်လှစ်ရခြင်း ရည်ရွယ်ချက်

ကွန်ပျူတာသိပ္ပံပညာရှင်များအနေဖြင့် ကွန်ပျူတာများသည် ထပ်ခါတလဲလဲ ပြုလုပ်ရသော အလုပ်များကို ကူညီဆောင်ရွက်ပေးရာတွင် အလွန်ကောင်းမွန်ကြောင်း ကျွန်ုပ်တို့ သိရှိကြသည်။ သို့သော်လည်း ဤအချက်သည် ကျွန်ုပ်တို့၏ ပရိုဂရမ်များ လုပ်ဆောင်စေလိုသည့် တွက်ချက်မှုများတွင်သာမက ကျွန်ုပ်တို့ကိုယ်တိုင် ကွန်ပျူတာကို *အသုံးပြုပုံ* တွင်လည်း အလားတူ သက်ရောက်ကြောင်း မကြာခဏဆိုသလို မေ့လျော့နေတတ်ကြသည်။ ကွန်ပျူတာနှင့် ပတ်သက်သည့် မည်သည့်ပြဿနာမဆို ဖြေရှင်းသည့်အခါ လုပ်ငန်းတွင်ကျယ်မှု ပိုမိုရှိစေရန်နှင့် ပိုမိုရှုပ်ထွေးသော ပြဿနာများကို ဖြေရှင်းနိုင်စေရန်အတွက် ကျယ်ပြန့်လှသော tools များကို လက်တစ်ကမ်းတွင် အသင့်ရရှိနိုင်ပါသည်။ သို့သော် ကျွန်ုပ်တို့အနက် အများအပြားသည် ထို tools များ၏ သေးငယ်သော အစိတ်အပိုင်းမျှကိုသာ အသုံးပြုကြပြီး အဆင်ပြေရုံမျှ အလွတ်ကျက်မှတ်ထားသော ကွန်ပျူတာ command အနည်းငယ်ကိုသာ သိရှိကြကာ အခက်အခဲတွေ့သည့်အခါ အင်တာနက်မှ command များကို မစဉ်းစားမဆင်ခြင်ဘဲ ကူးယူဖော်ပြ (copy-paste) တတ်ကြသည်။

ဤသင်တန်းသည် ထိုပြဿနာကို [ဖြေရှင်းရန်](https://missing-semester-my.github.io/about/) ကြိုးပမ်းမှုတစ်ခု ဖြစ်ပါသည်။

ကျွန်ုပ်တို့သည် သင်သိရှိပြီးသား tools များကို အထိရောက်ဆုံး အသုံးပြုနည်းကို သင်ကြားပေးလိုပြီး၊ သင်၏ toolbox ထဲသို့ ထည့်သွင်းရန် tools အသစ်များကို ပြသပေးကာ၊ မိမိကိုယ်တိုင် tools အသစ်များကို ပိုမိုရှာဖွေလေ့လာရန် (အချို့ကို ဖန်တီးရန်လည်း ဖြစ်နိုင်သည်) စိတ်အားထက်သန်မှု မြှင့်တင်ပေးနိုင်ရန် မျှော်လင့်ပါသည်။ ဤသည်မှာ ကွန်ပျူတာသိပ္ပံ သင်ရိုးညွှန်းတမ်း အများစုတွင် လိုအပ်နေသော ပျောက်ဆုံးနေသည့် စာသင်နှစ်ဝက် (The Missing Semester) ဖြစ်သည်ဟု ကျွန်ုပ်တို့ ယုံကြည်ပါသည်။

သင်တန်းစနစ်နှင့် ဖွဲ့စည်းပုံ

အမှတ်မပေးသော (not-for-credit) ဤသင်တန်းတွင် ၁ နာရီကြာ သင်ခန်းစာ ၉ ခု ပါဝင်ပြီး၊ တစ်ခုစီသည် [သီးခြားခေါင်းစဉ်တစ်ခု](https://missing-semester-my.github.io/2026/) ကို အဓိကထားထားသည်။ သင်ခန်းစာများသည် သီးခြားစီ လေ့လာနိုင်သော ခေါင်းစဉ်များ ဖြစ်ကြသော်လည်း သင်တန်းကာလ ကြာမြင့်လာသည်နှင့်အမျှ ယခင်သင်ခန်းစာများမှ အကြောင်းအရာများကို သင်ကျွမ်းကျင်ပြီးသားဖြစ်သည်ဟု ကျွန်ုပ်တို့ ယူဆသွားမည်ဖြစ်သည်။ ကျွန်ုပ်တို့သည် စာတွေ့မှတ်စုများကို အွန်လိုင်းတွင် တင်ပေးထားသော်လည်း စာသင်ချိန်အတွင်း သင်ကြားပြသသည့် အကြောင်းအရာများ (ဥပမာ- လက်တွေ့ပြသမှု demos များ) သည် စာတွေ့မှတ်စုများတွင် ပါဝင်ချင်မှ ပါဝင်မည်ဖြစ်သည်။ လွန်ခဲ့သော နှစ်များနည်းတူ သင်ခန်းစာများကို ဗီဒီယိုရိုက်ကူးပြီး ရိုက်ကူးချက်များကို [အွန်လိုင်း](https://www.youtube.com/@MissingSemester) တွင် တင်ဆက်ပေးသွားမည်ဖြစ်သည်။

၁ နာရီကြာ သင်ခန်းစာ အနည်းငယ်အတွင်း အကြောင်းအရာ အမြောက်အမြားကို လွှမ်းခြုံနိုင်ရန် ကြိုးစားထားသောကြောင့် သင်ခန်းစာများသည် တော်တော်လေး စိတ်အာရုံစိုက်ရန် လိုအပ်ပါသည်။ မိမိ၏ လေ့လာနိုင်သော နှုန်းအတိုင်း အကြောင်းအရာများနှင့် ရင်းနှီးကျွမ်းကျင်စေရန်အတွက် သင်ခန်းစာတစ်ခုစီတွင် သော့ချက်အချက်များကို လမ်းညွှန်ပေးသည့် လေ့ကျင့်ခန်းများ ပါဝင်ပါသည်။ ကျွန်ုပ်တို့သည် သီးသန့် အမေးအဖြေပြုလုပ်ရန် Office Hours သတ်မှတ်ထားခြင်း မရှိသော်လည်း [OSSU Discord](https://ossu.dev/#community) ၏ `#missing-semester-forum` တွင် မေးခွန်းများ မေးမြန်းရန် သို့မဟုတ် missing-semester@mit.edu သို့ အီးမေးလ် ပေးပို့ရန် တိုက်တွန်းပါသည်။

အချိန်ကန့်သတ်ချက်ကြောင့် tool တစ်ခုချင်းစီကို အပြည့်အဝသင်တန်းတစ်ခုကဲ့သို့ အသေးစိတ် အချက်အလက်များအတိုင်း လွှမ်းခြုံနိုင်မည် မဟုတ်ပါ။ ဖြစ်နိုင်သမျှ tool သို့မဟုတ် ခေါင်းစဉ်တစ်ခုကို ပိုမိုနက်ရှိုင်းစွာ လေ့လာနိုင်ရန် အရင်းအမြစ်များကို လမ်းညွှန်ပေးရန် ကြိုးစားသွားမည်ဖြစ်သော်လည်း အထူးစိတ်ဝင်စားသည့် အကြောင်းအရာတစ်ခုခု ရှိပါက ကျွန်ုပ်တို့ထံ ဆက်သွယ်၍ လမ်းညွှန်ချက်များ တောင်းခံရန် တွန့်ဆုတ်မနေပါနှင့်!

နောက်ဆုံးအနေဖြင့် သင်တန်းနှင့်ပတ်သက်၍ အကြံပြုချက်များရှိပါက missing-semester@mit.edu သို့ အီးမေးလ်မှတစ်ဆင့် ပေးပို့နိုင်ပါသည်။

ခေါင်းစဉ် ၁ - Shell

Shell ဆိုတာ ဘာလဲ။

ယနေ့ခေတ် ကွန်ပျူတာများတွင် ညွှန်ကြားချက်များ ပေးပို့ရန်အတွက် interface (မျက်နှာပြင်) အမျိုးမျိုး ရှိကြပါသည်။ ဆန်းသစ်သော graphical user interfaces၊ အသံဖြင့် ညွှန်ကြားသည့် interfaces၊ AR/VR နှင့် မကြာသေးမီက ပေါ်ပေါက်လာသော LLM များ ဖြစ်ကြသည်။ ၎င်းတို့သည် အသုံးပြုမှု အခြေအနေ ၈၀% အတွက် အဆင်ပြေ သော်လည်း လုပ်ဆောင်နိုင်စွမ်းတွင် အခြေခံအားဖြင့် ကန့်သတ်ချက်များ ရှိတတ်ကြသည် — မရှိသေးသော ခလုတ်တစ်ခုကို နှိပ်၍ မရနိုင်သလို၊ ပရိုဂရမ် ရေးဆွဲမထားသော အသံမိန်န့်ခွန်းကိုလည်း ပေး၍ မရနိုင်ပါ။ သင်၏ ကွန်ပျူတာက ပံ့ပိုးပေးထားသော tools များကို အပြည့်အဝ အသုံးချနိုင်ရန် ရှေးမှုမပျက် စာသားအခြေပြု interface ဖြစ်သည့် Shell သို့ ဆင်းသက် အသုံးပြုရမည် ဖြစ်သည်။

သင် အသုံးပြုနိုင်သည့် ပလပ်ဖောင်း တိုင်းနီးပါးတွင် ပုံစံတစ်မျိုးမျိုးဖြင့် Shell ပါဝင်ပြီး၊ ၎င်းတို့အနက် အများအပြားတွင် ရွေးချယ်စရာ Shell အမြောက်အမြား ရှိကြသည်။ အသေးစိတ် အချက်အလက်များတွင် ကွဲပြားနိုင်သော်လည်း အဓိက သဘောတရားအားဖြင့် အားလုံး နီးပါး တူညီကြသည် - ပရိုဂရမ်များကို စေခိုင်းခြင်း၊ ၎င်းတို့ထံ input များ ပေးပို့ခြင်းနှင့် ၎င်းတို့၏ output များကို ပုံစံတကျ စစ်ဆေးကြည့်ရှုခြင်းများကို ပြုလုပ်နိုင်စေပါသည်။

Shell *prompt* (ညွှန်ကြားချက်များ ရိုက်ထည့်နိုင်သည့်နေရာ) ကို ဖွင့်ရန်အတွက် ရှေးဦးစွာ Shell ၏ မြင်ကွင်း interface ဖြစ်သော *terminal* တစ်ခု လိုအပ်ပါသည်။ သင်၏ စက်ပစ္စည်းတွင် တစ်ခု စက်ရုံထုတ် ပါဝင်ပြီးသား ဖြစ်နိုင်သည် သို့မဟုတ် အလွယ်တကူ တပ်ဆင် install လုပ်နိုင်သည်-

- **Linux:** `Ctrl + Alt + T` ကို နှိပ်ပါ (distribution အများစုတွင် လုပ်ဆောင်သည်)။ သို့မဟုတ် သင်၏ applications menu တွင် “Terminal” ဟု ရှာပါ။
- **Windows:** `Win + R` ကို နှိပ်ပါ၊ `cmd` သို့မဟုတ် `powershell` ဟု ရိုက်ထည့်ပြီး Enter နှိပ်ပါ။ သို့မဟုတ် Start menu တွင် “Terminal” သို့မဟုတ် “Command Prompt” ဟု ရှာပါ။
- **macOS:** `Cmd + Space` ကို နှိပ်၍ Spotlight ဖွင့်ပါ၊ “Terminal” ဟု ရိုက်ထည့်ပြီး Enter နှိပ်ပါ။ သို့မဟုတ် Applications → Utilities → Terminal တွင် ရှာပါ။

Linux နှင့် macOS တို့တွင် ၎င်းသည် Bourne Again SHell သို့မဟုတ် အတိုကောက် “bash” ကို ပုံမှန် အားဖြင့် ဖွင့်ပေးမည် ဖြစ်သည်။ ဤသည်မှာ အသုံးအများဆုံး Shell များထဲမှ တစ်ခုဖြစ်ပြီး၊ ၎င်း၏ syntax သည် အခြား Shell အများအပြားတွင် တွေ့ရမည် ဖြစ်သည့်အတိုင်း တူညီလှသည်။ Windows တွင်မူ သင်မည်သည့် command ဖွင့်ခဲ့သည်အပေါ်မူတည်၍ “batch” သို့မဟုတ် “powershell” Shell များဖြင့် ကြုံဆို

မည် ဖြစ်သည်။ ၎င်းတို့သည် Windows သီးသန့်ဖြစ်ပြီး ဤသင်တန်းတွင် အဓိက ထား သင်ကြားမည် မဟုတ်ပါ (သို့သော်လည်း ကျွန်ုပ်တို့ သင်ကြားမည့် အကြောင်းအရာ အများစုအတွက် တူညီသော သဘောတရားများ ရှိကြသည်)။ ထို့အစား **Windows Subsystem for Linux** (https://docs.microsoft.com/en-us/windows/wsl/) သို့မဟုတ် Linux virtual machine တစ်ခုကို အသုံးပြုရန် လိုအပ်မည် ဖြစ်သည်။

bash ထက် အသုံးပြုရ ပိုမို အဆင်ပြေစေသည့် မွမ်းမံမှုများ ပါဝင်သော အခြား Shell များလည်း ရှိကြသည် (fish နှင့် zsh တို့မှာ အသုံးအများဆုံးထဲတွင် ပါဝင်သည်)။ ၎င်းတို့သည် လူကြိုက်များသော်လည်း (ဆရာ အားလုံးလည်း တစ်ခုခုကို အသုံးပြုကြသည်) bash ကဲ့သို့ နေရာတိုင်းတွင် မရှိနိုင်သကဲ့သို့ တူညီသော သဘောတရားများပေါ်တွင် မှီခိုနေသဖြင့် ဤသင်ခန်းစာတွင် ၎င်းတို့ကို အဓိကထားမည် မဟုတ်ပါ။

ဒါကို ဘာကြောင့် စိတ်ဝင်စားသင့်သလဲ။

Shell သည် “မောက်စ်ဖြင့် လိုက်ကလစ်နှိပ်ခြင်း” ထက် (ပုံမှန်အားဖြင့်) ပိုမို မြန်ဆန်ရုံသာမက မည်သည့် graphical ပရိုဂရမ်တစ်ခုတွင်မှ အလွယ်တကူ ရှာမတွေ့နိုင်သော ဖော်ပြနိုင်စွမ်းအား ပါဝင်သည်။ နောက်ပိုင်းတွင် တွေ့မြင်ရမည့်အတိုင်း Shell သည် ပရိုဂရမ်များကို တီထွင်ဖန်တီးမှုရှိသော နည်းလမ်းများဖြင့် *ပေါင်းစပ်* ပြီး မည်သည့် လုပ်ငန်းမဆို အလိုအလျောက် ဆောင်ရွက်နိုင်စေသည့် စွမ်းရည်ကို ပေးစွမ်းသည်။

Shell အသုံးပြုပုံကို ကျွမ်းကျင်စွာ သိရှိခြင်းသည် Open-source ဆော့ဖ်ဝဲလ် ကမ္ဘာကို လေ့လာစုံစမ်းရာတွင် လည်းကောင်း (၎င်းတို့တွင် Shell ကို သုံးရသည့် တပ်ဆင်မှု ညွှန်ကြားချက်များ ပါဝင်လေ့ရှိသည်)၊ သင်၏ ဆော့ဖ်ဝဲလ် ပရောဂျက်များအတွက် Continuous Integration တည်ဆောက်ရာတွင်လည်းကောင်း (**Code Quality သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/code-quality/) တွင် ဖော်ပြထားသည့်အတိုင်း)၊ အခြား ပရိုဂရမ်များ ထိခိုက်ပျက်စီးသည့်အခါ အမှားများကို ရှာဖွေ ပြင်ဆင်ရာတွင်လည်းကောင်း အလွန် အသုံးဝင်ပါသည်။

Shell တွင် လမ်းကြောင်းရှာဖွေ သွားလာခြင်း

Terminal ကို စတင်ဖွင့်လိုက်သည့်အခါ ဤကဲ့သို့ တွေ့ရလေ့ရှိသော *prompt* တစ်ခုကို မြင်တွေ့ရမည် ဖြစ်သည်-

```
missing:~$
```

ဤသည်မှာ Shell ၏ အဓိက စာသား interface ဖြစ်သည်။ သင်သည် `missing` ဟုခေါ်သော စက်ထဲတွင် ရောက်ရှိနေပြီး၊ သင်၏ “လက်ရှိ အလုပ်လုပ်နေသည့် ဖိုဒါလမ်းကြောင်း (current working directory)” သို့မဟုတ် လက်ရှိ ရောက်ရှိနေသည့် နေရာမှာ `~` (“home” ၏ အတိုကောက်) ဖြစ်ကြောင်း ဖော်ပြသည်။ `$` က သင်သည် root အသုံးပြုသူ မဟုတ်ကြောင်း ဖော်ပြသည် (ထိုအကြောင်းကို နောက်မှ အသေးစိတ်

ပြောပါမည်)။ ဤ prompt တွင် သင်သည် *command* တစ်ခု ရိုက်ထည့်နိုင်ပြီး ထို command ကို Shell မှ ဘာသာပြန် လုပ်ဆောင်ပေးမည် ဖြစ်သည်။ အခြေခံအကျဆုံး command မှာ ပရိုဂရမ်တစ်ခုကို စေခိုင်း run ခြင်း ဖြစ်သည်-

```
missing:~$ date
Fri 10 Jan 2020 11:49:31 AM EST
missing:~$
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် `date` ပရိုဂရမ်ကို run ခဲ့ပြီး၊ ၎င်းသည် (မျှော်လင့်ထားသည့်အတိုင်း) လက်ရှိ ရက်စွဲနှင့် အချိန်ကို ရိုက်နှိပ်ပြသပေးသည်။ ထို့နောက် Shell က ကျွန်ုပ်တို့အား နောက်ထပ် run မည့် command တစ်ခု ရိုက်ထည့်ရန် တောင်းဆိုသည်။ ကျွန်ုပ်တို့သည် command တစ်ခုကို *arguments* (အချက်အလက်များ) နှင့်အတူလည်း run နိုင်သည်။

```
missing:~$ echo hello
hello
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် `echo` ပရိုဂရမ်ကို `hello` ဆိုသည့် argument နှင့်အတူ run ရန် Shell အား ညွှန်ကြားခဲ့သည်။ `echo` ပရိုဂရမ်သည် ၎င်းထံ ပေးပို့လိုက်သော argument များကို ရိုးရှင်းစွာ ပြန်လည် ရိုက်နှိပ်ပေးခြင်း ဖြစ်သည်။ Shell သည် command ကို ဟာကွက် (whitespace) များဖြင့် ပိုင်းခြား၍ စစ်ဆေးပြီး ပထမစကားလုံးဖြင့် ညွှန်ပြသော ပရိုဂရမ်ကို run ကာ နောက်ဆက်တွဲ စကားလုံးများကို ပရိုဂရမ် အသုံးပြုနိုင်သည့် argument များအဖြစ် ပေးပို့သည်။ အကယ်၍ သင်သည် ကွက်လပ်များ သို့မဟုတ် အထူးသင်္ကေတများ ပါဝင်သော argument တစ်ခု (ဥပမာ- “My Photos” ဟု အမည်ရသော ဖိုဒါ) ကို ပေးပို့လိုပါက argument ကို `'` သို့မဟုတ် `"` ဖြင့် သတ်မှတ်နိုင်သည် (`"My Photos"`)၊ သို့မဟုတ် သက်ဆိုင်ရာ သင်္ကေတများကို `\` ဖြင့် escape လုပ်နိုင်သည် (`My\ Photos`)။

စတင်လေ့လာချိန်တွင် အရေးကြီးဆုံး command မှာ “manual” ၏ အတိုကောက် ဖြစ်သော `man` command ဖြစ်ပေလိမ့်မည်။ `man` ပရိုဂရမ်သည် အခြားအရာများအပြင် သင်၏ system ပေါ်ရှိ မည်သည့် command ၏ အသေးစိတ် အချက်အလက်များကိုမဆို ရှာဖွေ ကြည့်ရှုနိုင်စေသည်။ ဥပမာအားဖြင့် `man date` ကို run ပါက `date` ဆိုသည်မှာ ဘာလဲဆိုသည်နှင့် ၎င်း၏ လုပ်ဆောင်ချက်ကို ပြောင်းလဲရန် ပေးပို့နိုင်သည့် argument အမျိုးမျိုးကို ရှင်းပြပေးမည် ဖြစ်သည်။ command အများစုအတွက် `--help` ကို argument အဖြစ် ပေးပို့ခြင်းဖြင့်လည်း ကူညီမှု အတိုချုပ်ကို ရရှိနိုင်လေ့ ရှိသည်။

> `man` အပြင် **`tldr`** (https://tldr.sh/) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ၊ ၎င်းသည် အသုံးများသော ဥပမာများကို terminal ထဲတွင်ပင် တိုက်ရိုက် ပြသပေးသောကြောင့် ဖြစ်သည်။ LLM များသည်လည်း command များ မည်သို့ အလုပ်လုပ်ကြောင်းနှင့် မိမိ လိုချင်သည့် ရလဒ် ရရှိရန် မည်သို့ ခေါ်ယူသုံးစွဲရကြောင်း ရှင်းပြရာတွင် အလွန် ကောင်းမွန်ကြသည်။

`man` ပြီးလျှင် လေ့လာရန် အရေးကြီးဆုံး command မှာ “change directory” ၏ အတိုကောက်ဖြစ်သော `cd` ဖြစ်သည်။ ဤ command သည် သီးခြား ပရိုဂရမ် မဟုတ်ဘဲ Shell ထဲတွင် တိုက်ရိုက် ပါဝင်ပြီးသား (built-in) ဖြစ်သည် (ဥပမာ- `which cd` ကို run ပါက “no cd found” ဟု ဖော်ပြလိမ့်မည်)။ သင်သည် လမ်းကြောင်း path တစ်ခု ပေးပို့လိုက်ပါက ထို path သည် သင်၏ လက်ရှိ အလုပ်လုပ်နေသော ဖိုဒါလမ်းကြောင်း ဖြစ်လာမည် ဖြစ်သည်။ လက်ရှိ ရောက်ရှိနေသော ဖိုဒါလမ်းကြောင်းကို Shell prompt တွင် လည်း တွေ့မြင်ရမည် ဖြစ်သည်-

```
missing:~$ cd /bin
missing:/bin$ cd /
missing:/$ cd ~
missing:~$
```

> Shell တွင် auto-completion ပါဝင်သောကြောင့် `<TAB>` ကို နှိပ်ခြင်းဖြင့် လမ်းကြောင်းများကို ပိုမို မြန်ဆန်စွာ ရိုက်ထည့်နိုင်သည်ကို သတိပြုပါ!

အခြားနေရာ သီးသန့် မသတ်မှတ်ထားပါက command အများစုသည် လက်ရှိ အလုပ်လုပ်နေသော ဖိုဒါလမ်းကြောင်းပေါ်တွင် လုပ်ဆောင်ကြသည်။ သင် မည်သည့်နေရာတွင် ရောက်ရှိနေသည်ကို မသေချာပါက `pwd` ကို run နိုင်သည် သို့မဟုတ် `$PWD` environment variable ကို ရိုက်နှိပ်ကြည့်နိုင်သည် (`echo $PWD` ဖြင့်)၊ ၎င်းတို့ နှစ်ခုလုံးသည် လက်ရှိ ရောက်ရှိနေသော ဖိုဒါလမ်းကြောင်းကို ပြသပေးမည် ဖြစ်သည်။

လက်ရှိ အလုပ်လုပ်နေသော ဖိုဒါလမ်းကြောင်းသည် *relative* (နှိုင်းယှဉ်) paths များကို အသုံးပြုနိုင်စေသည့်အတွက်လည်း အလွန် အဆင်ပြေသည်။ ယခုအထိ ကျွန်ုပ်တို့ တွေ့ခဲ့ရသော paths အားလုံးသည် *absolute* (အပြည့်အစုံ) paths များ ဖြစ်ကြသည် — ၎င်းတို့သည် `/` ဖြင့် စတင်ပြီး file system ၏ root (`/`) မှစ၍ မည်သည့် နေရာသို့မဆို သွားရောက်နိုင်သည့် ဖိုဒါလမ်းကြောင်း အပြည့်အစုံကို ပေးသည်။ လက်တွေ့တွင် relative paths များကို ပိုမို သုံးလေ့ ရှိကြသည်၊ အကြောင်းမှာ ၎င်းတို့သည် လက်ရှိ ဖိုဒါလမ်းကြောင်းအပေါ် မူတည်၍ သတ်မှတ်သောကြောင့် ဖြစ်သည်။ Relative path တစ်ခုတွင် (`/` ဖြင့် စမ

ထားသော မည်သည့် path မဆို) ပထမဆုံး လမ်းကြောင်း အစိတ်အပိုင်းကို လက်ရှိ ဖိုဒါထဲတွင် ရှာဖွေပြီး နောက်ဆက်တွဲ အစိတ်အပိုင်းများကို ပုံမှန်အတိုင်း ဆက်လက် သွားရောက်သည်။ ဥပမာအားဖြင့်-

```
missing:~$ cd /
missing:/$ cd bin
missing:/bin$
```

ဖိုဒါတိုင်းတွင် ရှိနေသော "အထူး" အစိတ်အပိုင်း နှစ်ခုလည်း ရှိသည်- `.` နှင့် `..` တို့ ဖြစ်ကြသည်။ `.` မှာ "ဤလက်ရှိ ဖိုဒါ" ဖြစ်ပြီး `..` မှာ "အထက်ဖိုဒါ (parent directory)" ဖြစ်သည်။ ထို့ကြောင့်-

```
missing:~$ cd /
missing:/$ cd bin/../bin/../bin/./././../bin/..
missing:/$
```

command argument မည်သည့်အရာအတွက်မဆို absolute နှင့် relative paths များကို အပြန်အလှန် လဲလှယ် အသုံးပြုနိုင်လေ့ ရှိသည်၊ relative path ကို သုံးသည့်အခါ မိမိ၏ လက်ရှိ ရောက်ရှိနေသော ဖိုဒါ လမ်းကြောင်း မည်သည်ဖြစ်ကြောင်းကိုသာ သတိရပါ။

> `cd` ဖြင့် သွားလာခြင်းကို မြန်ဆန်စေရန် **`zoxide`** (https://github.com/ajeetdsouza/zoxide) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ — `z` သည် သင် မကြာခဏ သွားရောက်လေ့ရှိသော လမ်းကြောင်းများ ကို မှတ်သားထားပြီး အနည်းငယ်မျှ ရိုက်ထည့်ရုံဖြင့် ဝင်ရောက်နိုင်စေသည်။

Shell တွင် မည်သည့် tools များ ရရှိနိုင်သနည်း။

သို့သော် Shell သည် `date` သို့မဟုတ် `echo` ကဲ့သို့သော ပရိုဂရမ်များကို မည်သို့ ရှာဖွေရမည်ကို မည်သို့ သိရှိသနည်း။ Shell အား command တစ်ခုကို run ရန် ခိုင်းစေသည့်အခါ ၎င်းသည် command ပေးလိုက်သည့် အခါ မည်သည့် ဖိုဒါများတွင် ရှာဖွေရမည်ကို စာရင်းပြုစုထားသော `$PATH` ဟုခေါ်သည့် *environment variable* ကို တိုင်ပင်စစ်ဆေးသည်-

```
missing:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
missing:~$ which echo
/bin/echo
missing:~$ /bin/echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

ကျွန်ုပ်တို့သည် `echo` command ကို run သည့်အခါ Shell သည် `echo` ပရိုဂရမ်ကို စေခိုင်းရမည် ဖြစ်ကြောင်း သိရှိပြီး `$PATH` ထဲရှိ `:` ဖြင့် ပိုင်းခြားထားသော ဖိုဒါစာရင်းများတွင် ထိုအမည်ရှိသော file ကို ရှာဖွေသည်။ တွေ့ရှိပါက run ပေးသည် (file သည် *executable* မောင်းနှင်နိုင်သော file ဖြစ်သည်ဟု ယူဆပါသည်၊ ထိုအကြောင်းကို နောက်မှ အသေးစိတ် ပြောပါမည်)။ `which` ပရိုဂရမ်ကို အသုံးပြု၍ ပေးထားသော ပရိုဂရမ်အမည်အတွက် မည်သည့် file ကို မောင်းနှင်ရမည်ကို ရှာဖွေနိုင်သည်။ ကျွန်ုပ်တို့ မောင်းနှင်လိုသော file ၏ *path* လမ်းကြောင်းကို တိုက်ရိုက် ပေးပို့ခြင်းဖြင့် `$PATH` ကို သုံးစရာမလိုဘဲ တိုက်ရိုက် run နိုင်သည်။

ဤသည်မှာ Shell တွင် ကျွန်ုပ်တို့ run နိုင်သော ပရိုဂရမ်များ *အားလုံး* ကို မည်သို့ သိရှိနိုင်သနည်းဆိုသည် အတွက် အရိပ်အမြွက် ပေးသည်- `$PATH` ပေါ်ရှိ ဖိုဒါများအားလုံး၏ ပါဝင်သည့် အရာများကို စာရင်းထုတ်ကြည့်ခြင်း ဖြင့် ဖြစ်သည်။ file များကို စာရင်းထုတ်ပေးသော `ls` ပရိုဂရမ်သို့ ဖိုဒါ path တစ်ခု ပေးပို့ခြင်းဖြင့် ဤသို့ ပြုလုပ်နိုင်သည်-

```
missing:~$ ls /bin
```

> ပိုမို အဆင်ပြေသော `ls` အတွေ့အကြုံအတွက် **`eza`** (https://eza.rocks/) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ။

ကွန်ပျူတာ အများစုတွင် ဤသို့ပြုလုပ်ပါက ပရိုဂရမ် အမြောက်အများကို ပြသပေးမည် ဖြစ်သော်လည်း ဤနေရာတွင် အရေးကြီးဆုံး အချို့ကိုသာ အဓိကထားသွားပါမည်။ ပထမဦးစွာ ရိုးရှင်းသော အချို့မှာ-

- `cat file` - `file` ၏ ပါဝင်သောအရာများကို ပြသပေးသည်။
- `sort file` - `file` ၏ စာကြောင်းများကို အစီအစဉ်အတိုင်း စီပေးသည်။
- `uniq file` - `file` ထဲရှိ ဆက်တိုက် ထပ်နေသော စာကြောင်းများကို ဖယ်ရှားပေးသည်။
- `head file` နှင့် `tail file` - သက်ဆိုင်ရာ `file` ၏ ရှေ့ဆုံး နှင့် နောက်ဆုံး စာကြောင်း အနည်းငယ်ကို ပြသပေးသည်။

> စာသားများ အရောင်စုံ ပြသနိုင်ရန်နှင့် Scrolling ပြုလုပ်နိုင်ရန် `cat` အစား **`bat`** (https://github.com/sharkdp/bat) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ။

`file` ထဲတွင် `pattern` နှင့် ကိုက်ညီသော စာကြောင်းများကို ရှာဖွေပေးသည့် `grep pattern file` လည်း ရှိသည်။ ဤ command သည် အလွန် အသုံးဝင်သကဲ့သို့ မျှော်လင့်ထားသည်ထက် ပိုမိုကျယ်ပြန့်သော လုပ်ဆောင်ချက်များ ပါဝင်သဖြင့် ပိုမို အာရုံစိုက်ရန် ထိုက်တန်ပါသည်။ `pattern` ဆိုသည်မှာ အမှန်တကယ်တွင် အလွန် ရှုပ်ထွေးသော ပုံစံများကို ဖော်ပြနိုင်သည့် *regular expression* ဖြစ်သည် — ထိုအကြောင်းကို Code Quality သင်ခန်းစာ တွင် **လွှမ်းခြုံ ဖော်ပြသွားမည့် ဖြစ်သည်** (https://missing-semester-my.github.io/2026/code-quality/#regular-expressions)။ file တစ်ခုအစား ဖိုဒါတစ်ခုကိုလည်း သတ်မှတ်နိုင်ပြီး (သို့မဟုတ် `.` အတွက် ချန်လှပ်ထားနိုင်သည်) ဖိုဒါတစ်ခုအတွင်းရှိ file အားလုံးကို အဆင့်ဆင့် ရှာဖွေရန် `-r` ကို ပေးပို့နိုင်သည်။

> ပိုမို မြန်ဆန်ပြီး အသုံးပြုရ လွယ်ကူသော (သို့သော် ရွှေ့ပြောင်းအသုံးပြုရမှု အနည်းငယ် လျော့နည်းသော) အခြားရွေးချယ်စရာအဖြစ် `grep` အစား **`ripgrep`** (https://github.com/BurntSushi/ripgrep) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ။ `ripgrep` သည် ပုံမှန်အားဖြင့် လက်ရှိ အလုပ်လုပ်နေသော ဖိုဒါကိုလည်း အလိုအလျောက် အဆင့်ဆင့် ရှာဖွေပေးမည် ဖြစ်သည်!

အနည်းငယ် ပိုမို ရှုပ်ထွေးသော interface ရှိသည့် အလွန် အသုံးဝင်သော tools များလည်း ရှိသည်။ ထိုအထဲမှ ပထမဆုံးမှာ ပရိုဂရမ်ဆန်သော file အယ်ဒီတာ ဖြစ်သည့် `sed` ဖြစ်သည်။ ၎င်းတွင် file များကို အလိုအလျောက် ပြင်ဆင်ရန် ကိုယ်ပိုင် ပရိုဂရမ်မင်း ဘာသာစကား ပါဝင်သော်လည်း အသုံးအများဆုံး နည်းလမ်းမှာ-

```
missing:~$ sed -i 's/pattern/replacement/g' file
```

ဤသည်မှာ `file` ထဲရှိ `pattern` နေရာများ အားလုံးကို `replacement` ဖြင့် အစားထိုးပေးသည်။ `-i` က အစားထိုးမှုများကို inline တိုက်ရိုက် ပြုလုပ်စေလိုကြောင်း ပြသသည် (`file` ကို မပြင်ဘဲ အစားထိုးထားသော အကြောင်းအရာများကို ရိုက်နှိပ်ပြခြင်း မဟုတ်ပါ)။ `s/` မှာ အစားထိုးလိုကြောင်း sed ပရိုဂရမ်မင်း ဘာသာစကားတွင် ဖော်ပြသည့် နည်းလမ်းဖြစ်သည်။ `/` က pattern နှင့် replacement ကို ပိုင်းခြားပေးသည်။ နောက်ဆုံးရှိ `/g` က စာကြောင်းတစ်ကြောင်းစီရှိ ပထမဆုံး တစ်ခုတည်း မဟုတ်ဘဲ ပါဝင်သမျှ အားလုံးကို အစားထိုးလိုကြောင်း ပြသသည်။ `grep` နည်းတူ ဤနေရာမှ `pattern` သည် regular expression ဖြစ်ပြီး ကြီးမားသော ဖော်ပြနိုင်စွမ်းကို ပေးသည်။ Regular expression အစားထိုးမှုများသည်

`replacement` အား ကိုက်ညီသော pattern ၏ အစိတ်အပိုင်းများကို ပြန်လည် ရည်ညွှန်းနိုင်စေသည်၊ ထို အကြောင်း ဥပမာတစ်ခုကို ခဏအတွင်း တွေ့ရပါမည်။

နောက်တစ်ခုမှာ သတ်မှတ်ထားသော အခြေအနေများနှင့် ကိုက်ညီသော file များကို (အဆင့်ဆင့်) ရှာဖွေနိုင် သည့် `find` ဖြစ်သည်။ ဥပမာအားဖြင့်-

```
missing:~$ find ~/Downloads -type f -name "*.zip" -mtime +30
```

Download ဖိုဒါထဲရှိ ရက်ပေါင်း ၃၀ ထက် ဟောင်းနွမ်းသော ZIP file များကို ရှာဖွေပေးသည်။

```
missing:~$ find ~ -type f -size +100M -exec ls -lh {} \;
```

Home ဖိုဒါထဲရှိ 100M ထက် ကြီးသော file များကို ရှာဖွေ၍ စာရင်းထုတ်ပေးသည်။ `-exec` သည် တန်းလန်း `;` ဖြင့် ဆုံးသော *command* တစ်ခုကို ယူဆောင်ပြီး (ဟာကွက်ကွဲသို့ပင် escape လုပ်ရန် လိုအပ်သည်)၊ ထို နေရာတွင် `{}` ကို ကိုက်ညီသော file path တစ်ခုစီဖြင့် `find` က အစားထိုးပေးကြောင်း သတိပြုပါ။

```
missing:~$ find . -name "*.py" -exec grep -l "TODO" {} \;
```

TODO အကြောင်းအရာများ ပါရှိသော မည်သည့် `.py` file များကိုမဆို ရှာဖွေပေးသည်။

`find` ၏ syntax သည် အနည်းငယ် ခက်ခဲနိုင်သော်လည်း ၎င်းသည် မည်မျှ အသုံးဝင်ကြောင်း သဘောပေါက်နိုင်မည်ဟု မျှော်လင့်ပါသည်။

> `find` အစား ပိုမို အသုံးပြုရ လွယ်ကူသော အတွေ့အကြုံအတွက် **`fd`** (https://github.com/sharkdp/fd) ကို တပ်ဆင် အသုံးပြုရန် စဉ်းစားကြည့်ပါ (သို့သော် ရွှေ့ပြောင်းသုံးစွဲနိုင်မှု အနည်းငယ် လျော့နည်း သည်)။

နောက်ထပ် လေ့လာရမည်မှာ `sed` ကဲ့သို့ပင် ကိုယ်ပိုင် ပရိုဂရမ်မင်း ဘာသာစကား ပါရှိသော `awk` ဖြစ်သည်။ `sed` ကို file များ ပြင်ဆင်ရန် တည်ဆောက်ထားပြီး `awk` ကိုမူ ၎င်းတို့ကို စစ်ဆေး ဘာသာပြန် ရန် တည်ဆောက်ထားသည်။ `awk` ၏ အသုံးအများဆုံး နည်းလမ်းမှာ မှတ်တမ်း (စာကြောင်း) တိုင်း၏ အစိတ်အပိုင်း အချို့ကိုသာ ထုတ်ယူလိုသော ပုံမှန် syntax ပါရှိသည့် ဒေတာ file များ (CSV file များကဲ့သို့) အတွက် ဖြစ်သည်-

```
missing:~$ awk '{print $2}' file
```

`file` ၏ စာကြောင်းတိုင်းရှိ ဟာကွက်ဖြင့် ပိုင်းခြားထားသော ဒုတိယ ကော်လံကို ရိုက်နှိပ်ပေးသည်။ `-F,` ကို ထည့်သွင်းပါက ကော်မာဖြင့် ပိုင်းခြားထားသော ဒုတိယ ကော်လံကို ရိုက်နှိပ်ပေးမည် ဖြစ်သည်။ `awk` သည် စာကြောင်းများ စစ်ထုတ်ခြင်း၊ စုစုပေါင်း တွက်ချက်ခြင်း စသည်ဖြင့် ပိုမို ပြုလုပ်နိုင်ပါသည် — နမူနာ အဖြစ် လေ့ကျင့်ခန်းများကို ကြည့်ပါ။

ဤ tools များကို ပေါင်းစပ်လိုက်လျှင် ဤကဲ့သို့ ဆန်းသစ်သော အရာများကို ပြုလုပ်နိုင်သည်-

```
missing:~$ ssh myserver 'journalctl -u sshd -b-1 | grep "Disconnected from"' \
  | sed -E 's/.*Disconnected from .* user (.*) [^ ]+ port.*/\1/' \
  | sort | uniq -c \
  | sort -nk1,1 | tail -n10 \
  | awk '{print $2}' | paste -sd,
postgres,mysql,oracle,dell,ubuntu,inspur,test,admin,user,root
```

ဤသည်မှာ Remote server တစ်ခုမှ SSH logs များကို ရယူပြီး (`ssh` အကြောင်းကို နောက်သင်ခန်းစာတွင် ပိုမို ပြောပါမည်)၊ အဆက်အသွယ် ပြတ်တောက်မှု မက်ဆေ့ဂျ်များကို ရှာဖွေကာ၊ ထို မက်ဆေ့ဂျ်များမှ username ကို ထုတ်ယူ၍ အများဆုံး ပါဝင်သော သုံးစွဲသူ အမည် ၁၀ ခုကို ကော်မာခြားပြီး ရိုက်နှိပ်ပေးသည်။ command တစ်ခုတည်းဖြင့် ရရှိခြင်း ဖြစ်သည်! အဆင့်တစ်ခုစီကို အသေးစိတ် ခွဲခြားလေ့လာခြင်းကို လေ့ကျင့်ခန်းအဖြစ် ချန်လှပ်ထားပါမည်။

Shell ဘာသာစကား (bash)

ယခင် ဥပမာသည် သဘောတရား အသစ်တစ်ခုကို မိတ်ဆက်ပေးခဲ့သည်- pipes (`|`)။ ၎င်းတို့သည် ပရိုဂရမ် တစ်ခု၏ output ကို အခြား ပရိုဂရမ်တစ်ခု၏ input နှင့် ချိတ်ဆက်ပေးသည်။ command-line ပရိုဂရမ် အများစုသည် မည်သည့် `file` argument မျှ မပေးထားပါက ၎င်းတို့၏ "standard input" (ပုံမှန်အားဖြင့် သင် ရိုက်ထည့်သော စာသားများ ရောက်ရှိရာ) ပေါ်တွင် လုပ်ဆောင်သောကြောင့် ဤသို့ လုပ်ဆောင်နိုင်ခြင်း ဖြစ်သည်။ `|` သည် ၎င်း၏ ရှေ့ရှိ ပရိုဂရမ်မှ "standard output" (ပုံမှန်အားဖြင့် terminal တွင် ရိုက်နှိပ်ပြသ ရာ) ကို ယူပြီး `|` ၏ နောက်ရှိ ပရိုဂရမ်အတွက် standard input ဖြစ်စေသည်။ ဤသည်မှာ Shell ပရိုဂရမ် များကို *ပေါင်းစပ်* နိုင်စေပြီး Shell အား လုပ်ငန်းတွင်ကျယ်သော ဝန်းကျင်တစ်ခု ဖြစ်စေသည့် အချက်တစ်ခု ဖြစ်သည်!

အမှန်စင်စစ် Shell အများစုသည် Python သို့မဟုတ် Ruby ကဲ့သို့ပင် ပြည့်စုံသော ပရိုဂရမ်မင်း ဘာသာစကား ကို (bash ကဲ့သို့) အကောင်အထည်ဖော်ထားကြသည်။ ၎င်းတွင် variables၊ conditionals၊ loops နှင့် functions

များ ပါဝင်သည်။ သင်၏ Shell တွင် command များ run သည့်အခါ အမှန်တကယ်အားဖြင့် သင်သည် Shell မှ ဘာသာပြန်မည့် ကုဒ် အနည်းငယ်ကို ရေးသားနေခြင်း ဖြစ်သည်။ ယနေ့တွင် bash တစ်ခုလုံးကို သင်ကြားမည် မဟုတ်သော်လည်း အထူး အသုံးဝင်မည့် အစိတ်အပိုင်း အချို့ကို ဖော်ပြပေးပါမည်-

ပထမဦးစွာ Redirects များ- `>file` သည် ပရိုဂရမ်တစ်ခု၏ standard output ကို ယူပြီး terminal သို့ ရိုက်နှိပ်မပြဘဲ `file` ထဲသို့ ရေးသားပေးသည်။ ဤသည်မှာ နောက်ပိုင်းတွင် ဓာတ်ခွဲစစ်ဆေးရန် ပိုမို လွယ်ကူစေသည်။ `>>file` သည် `file` ကို ထပ်ရေး (overwrite) ခြင်း မပြုဘဲ စာကြောင်းများ ထပ်ပေါင်းပေးမည် ဖြစ်သည်။ ပရိုဂရမ်အတွက် standard input အဖြစ် ကီးဘုတ်အစား `file` မှ ဖတ်ရှုရန် Shell အား ညွှန်ကြားသည့် `<file` လည်း ရှိသည်။

> ဤသည်မှာ `tee` ပရိုဂရမ်အကြောင်း ပြောပြရန် သင့်တော်သော အချိန်ဖြစ်ပါသည်။ `tee` သည် standard input ကို standard output သို့ ရိုက်နှိပ်ပေးမည်ဖြစ်သလို (`cat` ကဲ့သို့ပင်!) file ထဲသို့လည်း *ရေးသားပေးမည်* ဖြစ်သည်။ ထို့ကြောင့် `verbose cmd | tee verbose.log | grep CRITICAL` သည် terminal ကို သန့်ရှင်းစေပြီး ရလဒ်အပြည့်အစုံကို log file ထဲတွင် သိမ်းဆည်းပေးမည် ဖြစ်သည်!

နောက်တစ်ခုမှာ Conditionals များ- `if command1; then command2; command3; fi` သည် `command1` ကို run မည်ဖြစ်ပြီး အမှား မရှိပါက `command2` နှင့် `command3` တို့ကို run မည်ဖြစ်သည်။ လိုအပ်ပါက `else` အခက်အလက်ကိုလည်း ထည့်သွင်းနိုင်သည်။ `command1` အဖြစ် သုံးလေ့ရှိသော အသုံးအများဆုံး command မှာ `test` command ဖြစ်ပြီး အတိုကောက်အားဖြင့် `[` ဟု ဖော်ပြလေ့ရှိကာ “file ရှိပါသလား” (`test -f file` / `[ -f file ]`) သို့မဟုတ် “string စာသား တူညီပါသလား” (`[ "$var" = "string" ]`) ကဲ့သို့ အခြေအနေများကို စစ်ဆေးပေးသည်။ bash တွင် ပိုမို “ဘေးကင်းသော” built-in `test` ပုံစံ ဖြစ်သည့် `[[ ]]` လည်း ရှိပြီး ကိုးကားချက် quoting ဆိုင်ရာ ပုံမှန်မဟုတ်သော မူမမှန်မှု များ ပိုမို နည်းပါးသည်။

Bash တွင် `while` နှင့် `for` ဆိုသည့် Loop ပုံစံ နှစ်မျိုး ပါဝင်သည်။ `while command1; do command2; command3; done` သည် `command1` အမှားမရှိသမျှ ကာလပတ်လုံး တစ်ခုလုံးကို ထပ်ခါထပ်ခါ run နေမည်မှလွဲ၍ သက်ဆိုင်ရာ `if` command ကဲ့သို့ လုပ်ဆောင်သည်။ `for varname in a b c d; do command; done` သည် `command` ကို လေးကြိမ် run ပြီး ကြိမ်တိုင်းတွင် `$varname` ကို `a` ၊ `b` ၊ `c` ၊ `d` တို့ဖြင့် သတ်မှတ်သည်။ အရာများကို တိုက်ရိုက် ရေးသားခြင်းထက် ဤကဲ့သို့သော “command substitution” ကို အသုံးပြုလေ့ ရှိကြသည်-

```
for i in $(seq 1 10); do
```

ဤသည်မှာ `seq 1 10` command ကို run ပြီး (၁ မှ ၁၀ အထိ ကိန်းဂဏန်းများကို ရိုက်နှိပ်ပေးသည်) ထို command ၏ output ဖြင့် `$()` တစ်ခုလုံးကို အစားထိုးပေးသောကြောင့် အကြိမ် ၁၀ ကြိမ် လည်ပတ်သည့် for loop ကို ရရှိစေသည်။ ရှေးကျသော ကုဒ်များတွင် `$()` အစား backticks များကို (`for i in `seq 1 10`; do` ကဲ့သို့) တွေ့ရတတ်သော်လည်း `$()` ကို ထပ်ဆင့် (nested) အသုံးပြုနိုင်သောကြောင့် ၎င်းကိုသာ အဓိက သုံးစွဲသင့်ပါသည်။

ရှည်လျားသော shell scripts များကို သင်၏ prompt ထဲတွင် တိုက်ရိုက် ရေးသားနိုင်သော်လည်း `.sh` file တစ်ခုထဲသို့ ရေးသားလေ့ ရှိကြသည်။ ဥပမာအားဖြင့် မအောင်မြင်မချင်း ပရိုဂရမ်တစ်ခုကို loop ပတ်၍ run ပြီး မအောင်မြင်သော ပုံစံ၏ output ကိုသာ ရိုက်နှိပ်ပြသကာ အနောက်ကွယ်တွင် CPU အား ဝန်ပိစေမည့် script တစ်ခု ဖြစ်ပါသည် (မတည်ငြိမ်သော flaky tests များကို ပြန်လည် စမ်းသပ်ရာတွင် အသုံးဝင်သည်)-

```
#!/bin/bash
set -euo pipefail

# Start CPU stress in background
stress --cpu 8 &
STRESS_PID=$!

# Setup log file
LOGFILE="test_runs_$(date +%s).log"
echo "Logging to $LOGFILE"

# Run tests until one fails
RUN=1
while cargo test my_test > "$LOGFILE" 2>&1; do
    echo "Run $RUN passed"
    ((RUN++))
done

# Cleanup and report
kill $STRESS_PID
echo "Test failed on run $RUN"
echo "Last 20 lines of output:"
tail -n 20 "$LOGFILE"
echo "Full log: $LOGFILE"
```

ဤနေရာတွင် အသစ်အဆန်း အချို့ ပါဝင်ပြီး ပရိုဂရမ်များကို ပြိုင်တူ run ရန် အနောက်ကွယ် အလုပ်များ (`&`)၊ ပိုမို ရှုပ်ထွေးသော **shell redirections** (https://www.gnu.org/software/bash/manual/html_node/Redirections.html) နှင့် **arithmetic expansion** (https://www.gnu.org/software/bash/manual/html_node/Arithmetic-Expansion.html) များ ကဲ့သို့ အသုံးဝင်သော shell command များ ရေးသားရာတွင် လေ့လာရန် အကြံပြုပါသည်။

ပရိုဂရမ်၏ ပထမ စာကြောင်း နှစ်ကြောင်းကို စိစစ်ကြည့်ရန် ထိုက်တန်ပါသည်။ ပထမ စာကြောင်းမှာ “shebang” ဖြစ်ပြီး shell scripts မဟုတ်သော အခြား file ၏ အပေါ်ဆုံးတွင်လည်း တွေ့ရမည် ဖြစ်သည်။ `#!/path` ဖြင့် စတင်သော file တစ်ခုကို run သည့်အခါ Shell သည် `/path` ရှိ ပရိုဂရမ်ကို စတင်ပြီး file ၏ ပါဝင်သည့် အရာများကို input အဖြစ် ပေးပို့သည်။ Shell script တစ်ခုတွင် ဤသည်မှာ shell script ကို `/bin/bash` သို့ ပေးပို့ခြင်းဖြစ်ပြီး Python script များကိုလည်း `/usr/bin/python` shebang စာကြောင်းဖြင့် ရေးသားနိုင်ပါသည်!

ဒုတိယ စာကြောင်းမှာ bash ကို ပိုမို “တင်းကျပ်” စေပြီး shell scripts ရေးသားရာတွင် ဖြစ်လေ့ရှိသော အမှားများကို လျော့ပါးစေမည့် နည်းလမ်း ဖြစ်သည်။ `set` သည် argument အမြောက်အများ ယူဆောင်နိုင်သော်လည်း အတိုချုပ်အားဖြင့်- `-e` သည် command တစ်ခုခု ပျက်စီးပါက script ကို စောစီးစွာ ထွက်ခွာစေသည်၊ `-u` သည် သတ်မှတ်မထားသော variables အသုံးပြုပါက စာသားအလွတ် အစား script ကို ထိခိုက်ရပ်တန့်စေသည်၊ `-o pipefail` သည် `|` တန်းစီဇယားထဲရှိ ပရိုဂရမ်များ ပျက်စီးပါက script တစ်ခုလုံးကို စောစီးစွာ ထွက်ခွာစေသည်။

> Shell programming သည် အခြား ပရိုဂရမ်မင်း ဘာသာစကားများကဲ့သို့ပင် နက်ရှိုင်းသော ခေါင်းစဉ် ဖြစ်သော်လည်း သတိပြုပါ- bash တွင် ပုံမှန်မဟုတ်သော အမှားထောင်ချောက်များ (gotchas) အမြောက်အများ ရှိနေပြီး ၎င်းတို့ကို စာရင်းပြုစုထားသော (https://mywiki.wooledge.org/BashPitfalls) ဝက်ဘ်ဆိုက်များပင် အများအပြား (https://tldp.org/LDP/abs/html/gotchas.html) ရှိကြသည်။ ၎င်းတို့ကို ရေးသားသည့်အခါ **shellcheck** (https://www.shellcheck.net/) ကို ထိရောက်စွာ အသုံးပြုရန် အလွန် အကြံပြုပါသည်။ LLM များသည်လည်း shell scripts များကို ရေးသားခြင်း၊ စစ်ဆေးခြင်းနှင့် bash ထက် ပိုမို ကြီးမားလာသောအခါ (စာကြောင်း ၁၀၀+) “အမှန်တကယ်” ပရိုဂရမ်မင်း ဘာသာစကား (Python ကဲ့သို့) သို့ ပြောင်းလဲပေးရာတွင် အလွန် ကောင်းမွန်ကြသည်။

နောက်ဆက်တွဲ လေ့လာရန်များ

ယခုအခါ အခြေခံ လုပ်ဆောင်ချက်များကို လုပ်ဆောင်နိုင်ရန်အတွက် Shell အသုံးပြုပုံကို ကျွမ်းကျင်စွာ သိရှိပြီး ဖြစ်သည်။ စိတ်ဝင်စားဖွယ် file များကို ရှာဖွေနိုင်ပြီး ပရိုဂရမ် အများစု၏ အခြေခံ လုပ်ဆောင်ချက်များကို အသုံးပြုနိုင်မည် ဖြစ်သည်။ နောက်သင်ခန်းစာတွင် Shell နှင့် tools command-line ပရိုဂရမ်များ အသုံးပြု၍ ပိုမို ရှုပ်ထွေးသော လုပ်ငန်းများကို အလိုအလျောက် ဆောင်ရွက်နည်းများအကြောင်း ပြောပြသွားပါမည်။

လေ့ကျင့်ခန်းများ

ဤသင်တန်းရှိ သင်ခန်းစာ အားလုံးတွင် လေ့ကျင့်ခန်းများ ပါဝင်သည်။ အချို့မှာ သီးခြား လုပ်ဆောင်ရမည့် အလုပ်တစ်ခုကို ပေးထားပြီး အချို့မှာ "X နှင့် Y ပရိုဂရမ်များကို အသုံးပြုကြည့်ပါ" ကဲ့သို့ ပွင့်လင်းသော လေ့ကျင့်ခန်းများ ဖြစ်ကြသည်။ ၎င်းတို့ကို စမ်းသပ်ကြည့်ရန် အလွန် တိုက်တွန်းပါသည်။

ကျွန်ုပ်တို့သည် လေ့ကျင့်ခန်းများအတွက် အဖြေများကို ရေးသားထားခြင်း မရှိပါ။ တစ်ခုခုတွင် အခက်အခဲတွေ့ပါက **Discord** (https://ossu.dev/#community) ရှိ `#missing-semester-forum` တွင် တင်ပြနိုင်သည် သို့မဟုတ် သင် ကြိုးစားထားသည်များကို အီးမေးလ် ပေးပို့နိုင်ပြီး ကျွန်ုပ်တို့ ကူညီပေးပါမည်။ ဤ လေ့ကျင့်ခန်းများသည် LLM နှင့် စကားပြောရာတွင် ခေါင်းစဉ်ထဲသို့ အပြန်အလှန် လေ့လာနိုင်သည့် စတင်မှု prompts များအဖြစ်လည်း ကောင်းစွာ အလုပ်လုပ်ပါလိမ့်မည်။ ဤလေ့ကျင့်ခန်းများ၏ အစစ်အမှန် တန်ဖိုးမှာ အဖြေကိုယ်တိုင် မဟုတ်ဘဲ အဖြေကို ရှာဖွေတွေ့ရှိသည့် ခရီးစဉ် ဖြစ်သည်။ အဖြေသို့ ရောက်ရှိမည့် အတိုဆုံး လမ်းကြောင်းကိုသာ ရှာဖွေခြင်းထက် လေ့လာနေစဉ် ဆက်စပ်အကြောင်းအရာများကို လိုက်လေ့လာရန်နှင့် "ဘာကြောင့်လဲ" ဟု မေးခွန်းထုတ်ရန် တိုက်တွန်းပါသည်။

1. ဤသင်တန်းအတွက် Bash သို့မဟုတ် ZSH ကဲ့သို့သော Unix shell တစ်ခုကို အသုံးပြုရန် လိုအပ်ပါသည်။ အကယ်၍ သင်သည် Linux သို့မဟုတ် macOS တွင် ရှိပါက သီးသန့် ပြုလုပ်ရန် မလိုပါ။ အကယ်၍ သင်သည် Windows တွင် ရှိပါက cmd.exe သို့မဟုတ် PowerShell ကို မသုံးကြောင်း သေချာပါစေ။ Unix ပုံစံ command-line tools များကို အသုံးပြုရန် **Windows Subsystem for Linux** (https://docs.microsoft.com/en-us/windows/wsl/) သို့မဟုတ် Linux virtual machine တစ်ခုကို သုံးနိုင်သည်။ သင့်လျော်သော shell ကို run နေကြောင်း သေချာစေရန် `echo $SHELL` command ကို စမ်းကြည့်ပါ။ အကယ်၍ `/bin/bash` သို့မဟုတ် `/usr/bin/zsh` ကဲ့သို့ ဖော်ပြပါက မှန်ကန်သော ပရိုဂရမ်ကို run နေခြင်း ဖြစ်သည်။

2. `ls` ၏ `-l` flag သည် မည်သို့ လုပ်ဆောင်သနည်း။ `ls -l /` ကို run ပြီး output ကို စိစစ်ပါ။ စာကြောင်းတစ်ကြောင်းစီ၏ ပထမဆုံး စာလုံး ၁၀ လုံးသည် မည်သည့် အဓိပ္ပာယ်လဲ။ (အကူအညီ- `man ls`)

3. `find ~/Downloads -type f -name "*.zip" -mtime +30` command တွင် `*.zip` သည် "glob" ဖြစ်သည်။ Glob ဆိုသည်မှာ ဘာလဲ။ file အချို့ ပါဝင်သော စမ်းသပ် ဖိုဒါတစ်ခု ဖန်တီးပြီး `ls *.txt` ၊ `ls file?.txt` နှင့် `ls {a,b,c}.txt` ကဲ့သို့ ပုံစံများကို စမ်းသပ်ပါ။ Bash manual ရှိ **Pattern Matching** (https://www.gnu.org/software/bash/manual/html_node/Pattern-Matching.html) ကို ကြည့်ပါ။

4. `'single quotes'` ၊ `"double quotes"` နှင့် `$'ANSI quotes'` တို့၏ ကွဲပြားချက်မှာ ဘာလဲ။ တိုက်ရိုက် `$` ၊ `!` နှင့် စာကြောင်းအသစ် (newline) စာလုံး ပါဝင်သော စာသားကို ရိုက်နှိပ်ပြသည့်

command တစ်ခု ရေးပါ။ **Quoting** (https://www.gnu.org/software/bash/manual/html_node/Quoting.html) ကို ကြည့်ပါ။

5. Shell တွင် ပုံမှန် stream သုံးခု ရှိသည်- stdin (0)၊ stdout (1) နှင့် stderr (2)။ `ls /nonexistent /tmp` ကို run ပြီး stdout ကို file တစ်ခုသို့လည်းကောင်း၊ stderr ကို အခြား file တစ်ခုသို့လည်းကောင်း redirect ပြုလုပ်ပါ။ နှစ်ခုလုံးကို file တစ်ခုတည်းသို့ မည်သို့ redirect လုပ်မည်နည်း။ **Redirections** (https://www.gnu.org/software/bash/manual/html_node/Redirections.html) ကို ကြည့်ပါ။

6. `$?` တွင် နောက်ဆုံး command ၏ exit status ပါဝင်သည် (0 = အောင်မြင်မှု)။ `&&` သည် ယခင် command အောင်မြင်ပါကမှ နောက် command ကို run ပြီး၊ `||` သည် ယခင် command မအောင်မြင်ပါကမှ run မည်။ `/tmp/mydir` မရှိသေးပါကမှ ဖန်တီးမည့် command တစ်ကြောင်း ရေးပါ။ **Exit Status** (https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html) ကို ကြည့်ပါ။

7. `cd` သည် သီးခြား ပရိုဂရမ်တစ်ခု မဟုတ်ဘဲ Shell ထဲတွင် တိုက်ရိုက် built-in အဖြစ် အဘယ်ကြောင့် ပါဝင်ရသနည်း။ (အကူအညီ- child process သည် parent process တွင် မည်သည့်အရာ ပြောင်းလဲနိုင်သည်၊ မပြောင်းလဲနိုင်သည်ကို စဉ်းစားပါ။)

8. File အမည်ကို argument (`$1`) အဖြစ် ယူပြီး `test -f` သို့မဟုတ် `[ -f ... ]` သုံး၍ file ရှိမရှိ စစ်ဆေးသည့် script တစ်ခု ရေးပါ။ File ရှိမရှိအပေါ် မူတည်၍ မတူညီသော မက်ဆေ့ဂျ်များ ရိုက်နှိပ်ရမည်။ **Bash Conditional Expressions** (https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html) ကို ကြည့်ပါ။

9. ယခင် လေ့ကျင့်ခန်းမှ script ကို file ထဲသို့ သိမ်းဆည်းပါ (ဥပမာ- `check.sh`)။ `./check.sh somefile` ဖြင့် run ကြည့်ပါ။ ဘာဖြစ်သနည်း။ ယခု `chmod +x check.sh` ကို run ပြီး ထပ်မံ စမ်းကြည့်ပါ။ ဤအဆင့်သည် အဘယ်ကြောင့် လိုအပ်သနည်း။ (အကူအညီ- `chmod` မတိုင်မီနှင့် အပြီးရှိ `ls -l check.sh` ကို ကြည့်ပါ။)

10. Script ထဲရှိ `set` flags များတွင် `-x` ထည့်ပါက ဘာဖြစ်မည်နည်း။ ရိုးရှင်းသော script တစ်ခုဖြင့် စမ်းသပ်ကြည့်ပြီး output ကို လေ့လာပါ။ **The Set Builtin** (https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html) ကို ကြည့်ပါ။

11. File ကို ဒီနေ့ ရက်စွဲ ပါသော backup သို့ ကူးယူသည့် command တစ်ခု ရေးပါ (ဥပမာ- `notes.txt` → `notes_2026-01-12.txt`)။ (အကူအညီ- `$(date +%Y-%m-%d)`)။ **Command Substitution** (https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html) ကို ကြည့်ပါ။

12. Lecture ထဲမှ မတည်ငြိမ်သော test script ကို `cargo test my_test` အစား test command ကို argument အဖြစ် လက်ခံရန် ပြင်ဆင်ပါ။ (အကူအညီ- `$1` သို့မဟုတ် `$@`)။ **Special Parameters** (https://www.gnu.org/software/bash/manual/html_node/Special-Parameters.html) ကို ကြည့်ပါ။

13. Pipes ကို အသုံးပြု၍ သင့် home ဖိုဒါထဲရှိ အသုံးအများဆုံး file extensions ၅ ခုကို ရှာပါ။ (အကူအညီ- `find` ၊ `grep` သို့မဟုတ် `sed` သို့မဟုတ် `awk` ၊ `sort` ၊ `uniq -c` နှင့် `head` တို့ကို ပေါင်းစပ်ပါ။)

14. `xargs` သည် stdin မှ စာကြောင်းများကို command arguments သို့ ပြောင်းလဲပေးသည်။ `find` နှင့် `xargs` ကို အတူတကွ သုံး၍ (`find -exec` မဟုတ်ပါ) ဖိုဒါတစ်ခုထဲရှိ `.sh` file များအားလုံးကို ရှာပြီး `wc -l` ဖြင့် တစ်ခုစီ၏ စာကြောင်း အရေအတွက်ကို ရေတွက်ပါ။ အပိုဆောင်း- ဟာကွက်ပါသော file အမည်များကို ကိုင်တွယ်နိုင်အောင် ပြုလုပ်ပါ။ (အကူအညီ- `-print0` နှင့် `-0`)။ `man xargs` ကို ကြည့်ပါ။

15. `curl` ကို အသုံးပြု၍ သင်တန်း ဝက်ဘ်ဆိုက်၏ HTML ကို ရယူပါ (`https://missing.csail.mit.edu/`) ပြီးလျှင် သင်ခန်းစာ အရေအတွက် မည်မျှ စာရင်းပါဝင်ကြောင်း ရေတွက်ရန် `grep` သို့ pipe ပြုလုပ်ပါ။ (အကူအညီ- သင်ခန်းစာ တစ်ခုစီတွင် ပါဝင်သည့် ပုံစံကို ရှာပါ၊ တိုးတက်မှု output ကို ပိတ်ရန် `curl -s` ကို သုံးပါ။)

16. `jq` (https://jqlang.github.io/jq/) သည် JSON ဒေတာများကို ကိုင်တွယ်ရန် စွမ်းအားထက်မြက်သော tool ဖြစ်သည်။ `https://microsoftedge.github.io/Demos/json-dummy-data/64KB.json` တွင်ရှိသော နမူနာ ဒေတာကို `curl` ဖြင့် ရယူပြီး version 6 ထက် ကြီးသော လူများ၏ အမည်များကိုသာ ထုတ်ယူရန် `jq` ကို သုံးပါ။ (အကူအညီ- ပုံသဏ္ဌာန်ကို ကြည့်ရန် ပထမဦးစွာ `jq .` သို့ pipe ပြုလုပ်ပါ၊ ထို့နောက် `jq '.[] | select(...) | .name'` ကို စမ်းသပ်ပါ။)

17. `awk` သည် ကော်လံ တန်ဖိုးများပေါ်မူတည်၍ စာကြောင်းများကို စစ်ထုတ်နိုင်ပြီး output ကို ပြုပြင်နိုင်သည်။ ဥပမာအားဖြင့် `awk '$3 ~ /pattern/ {$4=""; print}'` သည် တတိယ ကော်လံ pattern နှင့် ကိုက်ညီသော စာကြောင်းများကိုသာ ရိုက်နှိပ်ပြီး စတုတ္ထ ကော်လံကို ချန်လှပ်ထားသည်။ ဒုတိယ ကော်လံ ၁၀၀ ထက် ကြီးသော စာကြောင်းများကိုသာ ရိုက်နှိပ်ပြီး ပထမနှင့် တတိယ ကော်လံများကို လဲလှယ်ပေးသည့် `awk` command တစ်ခု ရေးပါ။ `printf 'a 50 x\nb 150 y\nc 200 z\n'` ဖြင့် စမ်းသပ်ပါ။

18. Lecture ထဲမှ SSH log pipeline ကို အသေးစိတ် စိစစ်ပါ- အဆင့်တစ်ခုစီသည် မည်သို့ လုပ်ဆောင်သနည်း။ ထို့နောက် `~/.bash_history` (သို့မဟုတ် `~/.zsh_history`) မှ အသုံးအများဆုံး shell commands များကို ရှာရန် အလားတူတစ်ခု တည်ဆောက်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. Anish	https://anish.io/	
2. Jon	https://thesquareplanet.com/	
3. Jose	http://josejg.com/	
4. သင်တန်းစာသင်ပစ္စည်းများ	https://missing.csail.mit.edu/	
5. သင်ခန်းစာ ဗီဒီယိုမှတ်တမ်းများ	https://www.youtube.com/@MissingSemester	
6. ဖြေရှင်းရန်	https://missing-semester-my.github.io/about/	
7. သီးခြားခေါင်းစဉ်တစ်ခု	https://missing-semester-my.github.io/2026/	
8. OSSU Discord	https://ossu.dev/#community	
9. Windows Subsystem for Linux	https://docs.microsoft.com/en-us/windows/wsl/	
10. Code Quality သင်ခန်းစာ	https://missing-semester-my.github.io/2026/code-quality/	
11. `tldr`	https://tldr.sh/	
12. `zoxide`	https://github.com/ajeetdsouza/zoxide	
13. `eza`	https://eza.rocks/	
14. `bat`	https://github.com/sharkdp/bat	
15. လွှမ်းခြုံ ဖော်ပြသွားမည် ဖြစ်သည်	https://missing-semester-my.github.io/2026/code-quality/#regular-expressions	

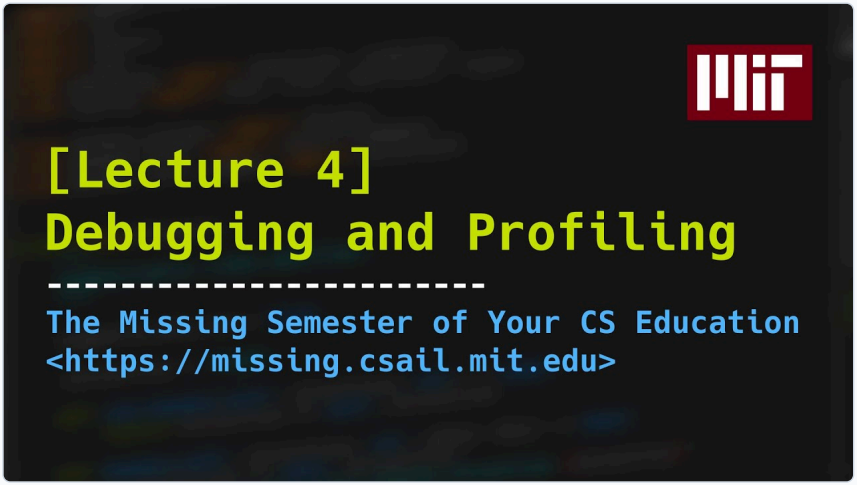
Resource / Description	URL	Micro QR
16. `ripgrep`	https://github.com/BurntSushi/ripgrep	
17. `fd`	https://github.com/sharkdp/fd	
18. shell redirections	https://www.gnu.org/software/bash/manual/html_node/Redirections.html	
19. arithmetic expansion	https://www.gnu.org/software/bash/manual/html_node/Arithmetic-Expansion.html	
20. စာရင်းပြုစုထားသော	https://mywiki.wooledge.org/BashPitfalls	
21. အများအပြား	https://tldp.org/LDP/abs/html/gotchas.html	
22. shellcheck	https://www.shellcheck.net/	
23. Pattern Matching	https://www.gnu.org/software/bash/manual/html_node/Pattern-Matching.html	
24. Quoting	https://www.gnu.org/software/bash/manual/html_node/Quoting.html	
25. Exit Status	https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html	
26. Bash Conditional Expressions	https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html	
27. The Set Builtin	https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html	
28. Command Substitution	https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html	
29. Special Parameters	https://www.gnu.org/software/bash/manual/html_node/Special-Parameters.html	
30. `jq`	https://jqlang.github.io/jq/	

Debugging နှင့် Profiling

အနှစ်ချုပ် - Logging နှင့် Debugger များကို အသုံးပြု၍ ပရိုဂရမ်များရှိ အမှားများကို ရှာဖွေပြင်ဆင်နည်း (debug လုပ်နည်း) နှင့် စွမ်းဆောင်ရည်မြင့်တင်ရန်အတွက် ကုဒ်များ၏ အလုပ်လုပ်ပုံကို တိုင်းတာစစ်ဆေးနည်း (profile လုပ်နည်း) တို့ကို လေ့လာသွားရမည် ဖြစ်သည်။

▶ LECTURE VIDEO REFERENCE

Debugging နှင့် Profiling



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=8VYT9TcUmKs

ပရိုဂရမ်းမင်းတွင် ရွေးချောင်စည်းမျဉ်းတစ်ခု ရှိသည် - ကုဒ်သည် သင်ဖြစ်စေချင်သည့်အတိုင်း အလုပ်လုပ်သည် မဟုတ်ဘဲ သင်ခိုင်းသည့်အတိုင်းသာ အလုပ်လုပ်သည်။ ထိုကွာဟချက်ကို ပေါင်းကူးပေးရန်မှာ အချို့အခါများတွင် တော်တော်ပင် ခက်ခဲသော စိန်ခေါ်မှုတစ်ခု ဖြစ်နိုင်သည်။ ဤပို့ချချက်တွင် အမှားပါနေသော ကုဒ်များ (buggy code) နှင့် အရင်းအမြစ်များကို အလွန်အမင်း သုံးစွဲနေသော ကုဒ်များ (resource-hungry code) ကို ဖြေရှင်းရာတွင် အသုံးဝင်သည့် နည်းလမ်းများဖြစ်ကြသော Debugging နှင့် Profiling တို့ကို လေ့လာသွားကြမည် ဖြစ်သည်။

Debugging

Printf Debugging နှင့် Logging

> “အထိရောက်ဆုံး Debugging Tool မှာ စေ့စပ်သေချာစွာ တွေးတောခြင်းနှင့် သင့်တော်သော နေရာများတွင် သတိထားထည့်သွင်းထားသည့် Print Statement များပင် ဖြစ်သည်” — Brian Kernighan, *Unix for Beginners*။

ပရိုဂရမ်တစ်ခုကို debug လုပ်ရန် ပထမဆုံး နည်းလမ်းမှာ သင်ပြဿနာတွေ့ရှိထားသော နေရာဝန်းကျင်တွင် print statement များကို ထည့်သွင်းပြီး ပြဿနာ၏ အဓိကအကြောင်းအရင်းကို နားလည်နိုင်ရန် လုံလောက်သော အချက်အလက်များ မရမချင်း အကြိမ်ကြိမ် စမ်းသပ်စစ်ဆေးသွားခြင်း ဖြစ်သည်။

ဒုတိယမြောက် နည်းလမ်းမှာ လိုအပ်မှ ချက်ချင်းထည့်ရေးသည့် ad hoc print statement များအစား သင့်ပရိုဂရမ်တွင် logging ကို အသုံးပြုခြင်း ဖြစ်သည်။ Logging ဆိုသည်မှာ အခြေခံအားဖြင့် “ပုံမိုသတိထား၍ print ထုတ်ခြင်း” ဖြစ်ပြီး အောက်ပါတို့ကဲ့သို့သော built-in ပါဝင်မှုများရှိသည့် logging framework တစ်ခုမှတစ်ဆင့် ပြုလုပ်လေ့ရှိသည် -

- log များကို (သို့မဟုတ် log များ၏ အစိတ်အပိုင်းများကို) အခြား output နေရာများသို့ လမ်းကြောင်းပြောင်း ပေးပို့နိုင်ခြင်း၊
- ပြင်းထန်မှုအဆင့်များ (severity levels - INFO, DEBUG, WARN, ERROR စသည်တို့) သတ်မှတ်နိုင်ပြီး ထိုအဆင့်များအတိုင်း output များကို စစ်ထုတ် (filter) နိုင်ခြင်း၊ နှင့်
- log entry များနှင့် သက်ဆိုင်သည့် ဒေတာများကို စနစ်တကျ ရေးသားနိုင်သော structured logging ကို ထောက်ပံ့ပေးခြင်းဖြစ်ပြီး ၎င်းတို့အား နောက်ပိုင်းတွင် ပိုမိုလွယ်ကူစွာ ထုတ်ယူစစ်ဆေးနိုင်ခြင်း။

Logging statement များကို ပရိုဂရမ် ရေးသားနေစဉ် ကတည်းက ကြိုတင်၍ ထည့်သွင်းထားလေ့ရှိသဖြင့် debug လုပ်ရန် လိုအပ်သော ဒေတာများသည် အဆင်သင့် ရှိနှင့်ပြီးသား ဖြစ်နေနိုင်ပေသည်! ထို့အပြင် print statement များကို သုံး၍ ပြဿနာကို ရှာဖွေဖြေရှင်းပြီးသည့်အခါ ထို print များကို ပြန်မဖျက်မီ သင့်တော်သော log statement များအဖြစ် ပြောင်းလဲပေးထားခြင်းသည် ပိုမို ထိရောက်မှု ရှိသည်။ ဤသို့ပြုလုပ်ခြင်းဖြင့် အနာဂတ်တွင် ဆင်တူသော bug များ ထပ်မံပေါ်ပေါက်လာပါက ကုဒ်ကို ထပ်မံပြင်ဆင်စရာ မလိုဘဲ လိုအပ်သော ရောဂါရှာဖွေရေး အချက်အလက်များ (diagnostic information) ကို အဆင်သင့် ရရှိနေမည် ဖြစ်သည်။

Third-party log များ- ပရိုဂရမ်အများအပြားသည် ၎င်းတို့ run ချိန်တွင် အချက်အလက်များကို ပိုမို ဖော်ပြရန် `-v` သို့မဟုတ် `--verbose` flag ကို ထောက်ပံ့ပေးထားကြသည်။ ၎င်းသည် ပေးထားသော command တစ်ခု အဘယ်ကြောင့် မအောင်မြင်ရသည်ကို ရှာဖွေရာတွင် အသုံးဝင်သည်။ အချို့ ပရိုဂရမ်များတွင် အသေးစိတ်အချက်အလက်များကို ပိုမိုရရှိရန် အဆိုပါ flag ကို အကြိမ်ကြိမ် ထပ်မံ ထည့်သွင်းခွင့်ပြုသည်။ Service များ (database များ၊ web server များ စသည်) ၏ ပြဿနာများကို debug လုပ်သည့်အခါ ၎င်းတို့၏ log များကို စစ်ဆေးပါ — Linux တွင် လော့ဂ်များကို `/var/log/` ၌ တွေ့ရလေ့ရှိသည်။ systemd service များ၏ log များကို ကြည့်ရှုရန် `journalctl -u <service>` ကို အသုံးပြုပါ။ Third-party library များအတွက် ၎င်းတို့သည် environment variable များ သို့မဟုတ် configuration များမှတစ်ဆင့် debug logging ကို ထောက်ပံ့ပေးခြင်း ရှိမရှိ စစ်ဆေးပါ။

Debugger များ

Print debugging သည် မည်သည့်အရာကို print ထုတ်ရမည်ကို သင်သိရှိပြီး သင့်ကုဒ်ကို လွယ်ကူစွာ ပြင်ဆင်၍ ပြန်လည် run နိုင်သည့်အခါများတွင် ကောင်းစွာ အလုပ်လုပ်သည်။ မည်သည့် အချက်အလက် လိုအပ်သည်ကို သေချာ မသိသေးသည့်အခါ၊ bug သည် ပြန်လည် ဖန်တီးရခက်ခဲသော အခြေအနေများတွင်မှ ပေါ်ပေါက်လာသည့်အခါ သို့မဟုတ် ပရိုဂရမ်ကို ပြင်ဆင်၍ ပြန်လည်စတင်ရန် ကုန်ကျစရိတ် များပြားသည့် အခါများတွင် (စတင်ချိန် ကြာမြင့်ခြင်း၊ ပြန်လည်ဖန်တီးရမည့် ရှုပ်ထွေးသော state များရှိခြင်း စသည်ဖြင့်) Debugger များသည် အလွန်တန်ဖိုးရှိလာသည်။

Debugger ဆိုသည်မှာ ပရိုဂရမ်တစ်ခု အလုပ်လုပ်နေစဉ် ၎င်း၏ လုပ်ဆောင်ချက်များနှင့် တိုက်ရိုက် ထိတွေ့ ဆောင်ရွက်နိုင်စေသည့် ပရိုဂရမ်များ ဖြစ်ကြပြီး အောက်ပါတို့အား ပြုလုပ်နိုင်စေသည် -

- သတ်မှတ်ထားသော လိုင်းတစ်ခုသို့ ရောက်ရှိသောအခါ ပရိုဂရမ် လုပ်ဆောင်ချက်ကို ခေတ္တရပ်တန့် (halt) ရန်။
- Instruction တစ်ခုချင်းစီအလိုက် တစ်ဆင့်ပြီးတစ်ဆင့် (step through) လုပ်ဆောင်သွားရန်။
- ပရိုဂရမ် crash ဖြစ်သွားပြီးနောက် variable များ၏ တန်ဖိုးများကို စစ်ဆေးရန်။
- ပေးထားသော အခြေအနေတစ်ခု ကိုက်ညီသည့်အခါ လုပ်ဆောင်ချက်ကို အခြေအနေပေါ်မူတည်၍ ခေတ္တ ရပ်တန့်ရန်။
- ထို့အပြင် အခြားသော အဆင့်မြင့် feature အများအပြားလည်း ပါဝင်သည်။

ပရိုဂရမ်းမင်း ဘာသာစကား အများစုသည် debugger ပုံစံတစ်မျိုးမျိုးကို ထောက်ပံ့ပေးထားကြသည် (သို့မဟုတ် အဆင်သင့် ပါဝင်လာကြသည်)။ အသုံးအဝင်ဆုံးများမှာ Native binary မည်သည့်အရာမဆို debug လုပ်နိုင်သည့် `gdb` (https://www.gnu.org/software/gdb/) (GNU Debugger) နှင့် `lldb` (https://lldb.llvm.org/) (LLVM Debugger) ကဲ့သို့သော **General-purpose debugger များ** ဖြစ်ကြသည်။ ဘာသာစကား အများအပြားတွင်လည်း runtime နှင့် ပိုမိုနီးကပ်စွာ ပေါင်းစပ်ထားသည့် **Language-specific debugger များ** ရှိကြသည် (Python ၏ pdb သို့မဟုတ် Java ၏ jdb ကဲ့သို့)။

`gdb` သည် C, C++, Rust နှင့် အခြားသော compiled ဘာသာစကားများအတွက် စံဖြစ်သော (de-facto standard) debugger တစ်ခုဖြစ်သည်။ ၎င်းသည် မည်သည့် process ကိုမဆို စစ်ဆေးနိုင်စေပြီး ၎င်း၏ လက်ရှိ machine state များဖြစ်သော register များ၊ stack၊ program counter နှင့် အခြားအရာများကို ရယူနိုင်စေသည်။

အသုံးဝင်သော GDB command အချို့ -

- `run` - ပရိုဂရမ်ကို စတင်ရန်
- `b {function}` သို့မဟုတ် `b {file}:{line}` - Breakpoint တစ်ခု သတ်မှတ်ရန်
- `c` - ပရိုဂရမ် လုပ်ဆောင်ချက်ကို ဆက်လက်လုပ်ဆောင်ရန်
- `step` / `next` / `finish` - Step in / step over / step out (ထဲသို့ဝင်ရန် / အပေါ်မှကျော်ရန် / အပြင်သို့ ထွက်ရန်)
- `p {variable}` - variable ၏ တန်ဖိုးကို print ထုတ်ရန်
- `bt` - Backtrace (call stack) ကို ဖော်ပြရန်
- `watch {expression}` - တန်ဖိုး ပြောင်းလဲသွားသည့်အခါ ရပ်တန့်ရန်

> Command prompt နှင့်အတူ source code ကို မျက်နှာပြင် ခွဲ၍ ကြည့်ရှုနိုင်ရန် GDB ၏ TUI mode (`gdb -tui` သို့မဟုတ် GDB အတွင်း၌ `Ctrl-x a` ကို နှိပ်ပါ) ကို အသုံးပြုရန် စဉ်းစားပါ။

Record-Replay Debugging

စိတ်ပျက်စရာ အကောင်းဆုံး bug အချို့မှာ *Heisenbug* များ ဖြစ်ကြသည် - ၎င်းတို့သည် သင့်အနေဖြင့် စောင့်ကြည့်လေ့လာရန် ကြိုးစားသောအခါ ပျောက်ကွယ်သွားသကဲ့သို့ ဖြစ်လေ့ရှိသော သို့မဟုတ် မူလမူပိုင် ပြဿနာ ဆောင်ရွက်ပုံများ ပြောင်းလဲသွားလေ့ရှိသော bug များ ဖြစ်ကြသည်။ Race condition များ၊ အချိန်အပေါ် မူတည်နေသော bug များ နှင့် စနစ်၏ အချို့သော အခြေအနေများတွင်မှ ပေါ်ပေါက်လာတတ်သော ပြဿနာများသည် ဤအမျိုးအစားတွင် ပါဝင်သည်။ ပရိုဂရမ်ကို ပြန်လည် run သောအခါ ကွဲပြားသော ပြဿနာ

ဆောင်ရွက်ချက်များကို ဖြစ်ပေါ်စေသောကြောင့် (ဥပမာ- print statement များသည် ကုဒ်ကို နှေးကွေးသွားစေနိုင်သဖြင့် race condition ထပ်မံဖြစ်ပွားတော့ခြင်းမျိုး) ထုံးတမ်းစဉ်လာ debugging နည်းလမ်းများသည် ဤနေရာတွင် အသုံးမဝင်လှပါ။

Record-replay debugging သည် ပရိုဂရမ်တစ်ခု၏ လုပ်ဆောင်ချက်ကို မှတ်တမ်းတင်ထားပြီး သင်လိုအပ်သလောက် အကြိမ်ကြိမ် မူလအတိုင်း အတိအကျ ပြန်လည်ပြသ (replay) နိုင်စေခြင်းဖြင့် ဤပြဿနာကို ဖြေရှင်းပေးသည်။ ထို့ထက်ပို၍ ကောင်းမွန်သည်မှာ သင်သည် မည်သည့်နေရာတွင် မှားယွင်းသွားခဲ့သည်ကို အတိအကျ ရှာဖွေနိုင်ရန် ပရိုဂရမ် လုပ်ဆောင်ချက်အား နောက်ပြန်လှည့် (*reverse*) ၍ စစ်ဆေးနိုင်သည်။

rr (https://rr-project.org/) သည် ပရိုဂရမ် လုပ်ဆောင်ချက်ကို မှတ်တမ်းတင်ပြီး အပြည့်အဝ debug လုပ်နိုင်သော စွမ်းရည်များဖြင့် မူလအတိုင်း အတိအကျ ပြန်လည်ပြသနိုင်စေသည့် Linux အတွက် စွမ်းဆောင်ရည်မြင့် tool တစ်ခုဖြစ်သည်။ ၎င်းသည် GDB နှင့် တွဲဖက် အလုပ်လုပ်သဖြင့် အင်တာဖေးစ်ကို သင်ရင်းနှီးပြီးသား ဖြစ်ပေလိမ့်မည်။

အခြေခံ အသုံးပြုပုံ -

```
# Record a program execution
rr record ./my_program

# Replay the recording (opens GDB)
rr replay
```

အံ့သြဖွယ်ရာ လုပ်ဆောင်ချက်များသည် replay ပြုလုပ်ချိန်တွင် ဖြစ်ပေါ်လာသည်။ ပရိုဂရမ်၏ လုပ်ဆောင်ချက်သည် မူလအတိုင်း အတိအကျ ဖြစ်သောကြောင့် **Reverse debugging** command များကို အသုံးပြုနိုင်သည် -

- `reverse-continue` (`rc`) - Breakpoint တစ်ခုသို့ မရောက်မချင်း နောက်ပြန် run ရန်
- `reverse-step` (`rs`) - နောက်သို့ တစ်လိုင်းချင်းစီ နောက်ပြန်သွားရန်
- `reverse-next` (`rn`) - Function call များကို ကျော်လွှန်၍ နောက်သို့ နောက်ပြန်သွားရန်
- `reverse-finish` - လက်ရှိ function အတွင်းသို့ မဝင်ရောက်မီအချိန်အထိ နောက်ပြန် run ရန်

၎င်းသည် debugging ပြုလုပ်ရန်အတွက် အလွန်ပင် စွမ်းအားထက်မြက်လှသည်။ ဥပမာ ပရိုဂရမ် crash ဖြစ်သွားသည်ဆိုပါစို့ — bug ၏ နေရာကို ခန့်မှန်း၍ breakpoint များ သတ်မှတ်နေမည့်အစား အောက်ပါအတိုင်း ပြုလုပ်နိုင်သည် -

1. Crash ဖြစ်သည့်နေရာအထိ Run ရန်
2. ပျက်စီးသွားသော state (corrupted state) ကို စစ်ဆေးရန်

3. ပျက်စီးသွားသော variable ပေါ်တွင် watchpoint တစ်ခု သတ်မှတ်ရန်
4. မည်သည့်နေရာတွင် အတိအကျ ပျက်စီးသွားခဲ့သည်ကို ရှာဖွေရန် `reverse-continue` ပြုလုပ်ရန်

rr ကို မည်သည့်အခါတွင် အသုံးပြုရမည်နည်း - - တစ်ခါတစ်ရံမှ ကျရှုံးတတ်သော Flaky test များအတွက် - Race condition များ နှင့် threading bug များအတွက် - ပြန်လည် ဖန်တီးရခက်ခဲသော crash များအတွက် - “အချိန်ကို နောက်ပြန်လှည့်နိုင်လျှင် ကောင်းမည်” ဟု သင်ဆုတောင်းမိစေသော bug မည်သည့်အရာမဆို အတွက်

> မှတ်ချက်- rr သည် Linux တွင်သာ အလုပ်လုပ်ပြီး hardware performance counter များ လိုအပ်ပါသည်။ အဆိုပါ counter များကို ထုတ်ဖော်ပြသထားခြင်းမရှိသော AWS EC2 instance အများစုကဲ့သို့သော VM များတွင် အလုပ်မလုပ်ပါ၊ ထို့အပြင် GPU access ကိုလည်း ထောက်ပံ့မပေးပါ။ macOS အတွက်မူ **Warpspeed** (https://warpspeed.dev/) ကို စစ်ဆေးကြည့်ပါ။

> **rr နှင့် Concurrency**- rr သည် လုပ်ဆောင်ချက်ကို အတိအကျ မှတ်တမ်းတင်ထားသောကြောင့် thread scheduling ကို စီရီယယ်လိုက် (serialize) ပြုလုပ်လိုက်သည်။ ဆိုလိုသည်မှာ အချို့သော race condition များသည် သီးခြား timing ပေါ်တွင် မူတည်ပါက rr အောက်တွင် ပေါ်ပေါက်လာမည် မဟုတ်ပါ။ သို့သော်လည်း race condition များကို debug လုပ်ရန် rr သည် အသုံးဝင်နေဆဲ ဖြစ်သည် — ကျရှုံးသွားသော run တစ်ခုကို ရယူလိုက်သည်နှင့် ၎င်းကို စိတ်ချယုံကြည်စွာ ပြန်လည်ပြသနိုင်သည် — သို့သော် အခါအားလျော်စွာ ပေါ်ပေါက်တတ်သော bug ကို ဖမ်းဆီးရန်အတွက် မှတ်တမ်းတင်ခြင်းကို အကြိမ်ကြိမ် ကြိုးစားရန် လိုအပ်နိုင်သည်။ Concurrency မပါဝင်သော bug များအတွက်မူ rr သည် အတောက်ပဆုံး စွမ်းဆောင်နိုင်သည် — အတိအကျ လုပ်ဆောင်ချက်ကို အမြဲတမ်း ပြန်လည် ဖန်တီးနိုင်ပြီး ပျက်စီးသွားသော နေရာကို လိုက်လံရှာဖွေရန် reverse debugging ကို အသုံးပြုနိုင်သည်။

System Call Tracing

အချို့အခါများတွင် သင့်ပရိုဂရမ်သည် operating system နှင့် မည်သို့ ချိတ်ဆက် ဆောင်ရွက်နေသည်ကို နားလည်ရန် လိုအပ်သည်။ ပရိုဂရမ်များသည် kernel ထံမှ ဝန်ဆောင်မှုများ (ဖိုင်များဖွင့်ခြင်း၊ memory သတ်မှတ်ပေးခြင်း၊ process များ ဖန်တီးခြင်း နှင့် အခြားအရာများ) ကို တောင်းဆိုရန်အတွက် **system call များ** (https://en.wikipedia.org/wiki/System_call) ကို ပြုလုပ်ကြသည်။ ဤ call များကို စောင့်ကြည့်ဆွဲထုတ်ခြင်း (tracing) ပြုလုပ်ခြင်းဖြင့် ပရိုဂရမ်တစ်ခု အဘယ်ကြောင့် ရပ်တန့် (hang) နေရသနည်း၊ မည်သည့်ဖိုင်များကို

ဝင်ရောက်ကြည့်ရှုရန် ကြိုးစားနေသနည်း သို့မဟုတ် မည်သည့်နေရာတွင် စောင့်ဆိုင်းရင်း အချိန်ကုန်လွန်နေသနည်း စသည်တို့ကို ဖော်ထုတ်ပေးနိုင်သည်။

strace (Linux) နှင့် dtruss (macOS)

`strace` (https://www.man7.org/linux/man-pages/man1/strace.1.html) သည် ပရိုဂရမ်တစ်ခု ပြုလုပ်သော system call တိုင်းကို စောင့်ကြည့်နိုင်စေသည် -

```
# Trace all system calls
strace ./my_program

# Trace only file-related calls
strace -e trace=file ./my_program

# Follow child processes (important for programs that start other programs)
strace -f ./my_program

# Trace a running process
strace -p <PID>

# Show timing information
strace -T ./my_program
```

> macOS နှင့် BSD တို့တွင် ဆင်တူသော လုပ်ဆောင်ချက်များအတွက် `dtruss` (https://www.manpagez.com/man/1/dtruss/) (`dtrace`) ကို အသုံးပြုပါ -

> `strace` အကြောင်း ပိုမိုနက်ရှိုင်းစွာ လေ့လာရန် Julia Evans ၏ ကောင်းမွန်လှသော **strace zine** (https://jvns.ca/strace-zine-unfolded.pdf) ကို စစ်ဆေးကြည့်ပါ။

bpftrace နှင့် eBPF

eBPF (https://ebpf.io/) (extended Berkeley Packet Filter) သည် kernel အတွင်း၌ စံသတ်မှတ်ထားသော သီးခြားပတ်ဝန်းကျင် (sandboxed) ဖြင့် ပရိုဂရမ်များ run နိုင်စေသည့် စွမ်းအားထက်မြက်သော Linux နည်းပညာတစ်ခု ဖြစ်သည်။ `bpftrace` (https://github.com/iovisor/bpftrace) သည် eBPF ပရိုဂရမ်များ ရေးသားရန် အတွက် အဆင့်မြင့် syntax တစ်ခုကို ပံ့ပိုးပေးသည်။ ၎င်းတို့သည် kernel အတွင်း run နေသော ပရိုဂရမ်များ

ဖြစ်ကြသဖြင့် ကြီးမားသော လုပ်ဆောင်နိုင်စွမ်း (awk နှင့် ဆင်တူသော အနည်းငယ် ရှုပ်ထွေးသည့် syntax ရှိသော်လည်း) ရှိကြသည်။ ၎င်းတို့အတွက် အတွေ့ရများဆုံး အသုံးပြုပုံမှာ စုစည်းချက်များ (အရည်အတွက် သို့မဟုတ် latency စာရင်းအင်းများကဲ့သို့) သို့မဟုတ် system call argument များကို စစ်ဆေးခြင်း (သို့မဟုတ် filter လုပ်ခြင်း) အပါအဝင် မည်သည့် system call များ ခေါ်ယူသုံးစွဲနေသည်ကို စုံစမ်းစစ်ဆေးခြင်း ဖြစ်သည်။

```
# Trace file opens system-wide (prints immediately)
sudo bpftrace -e 'tracepoint:syscalls:sys_enter_openat { printf("%s %s\n", com
m, str(args->filename)); }'

# Count system calls by name (prints summary on Ctrl-C)
sudo bpftrace -e 'tracepoint:syscalls:sys_enter_* { @[probe] = count(); }'
```

သို့သော်လည်း disk စနစ် လုပ်ဆောင်ချက်များ၏ latency ပျံ့နှံ့မှုကို print ထုတ်ပေးသော `biosnoop` သို့မဟုတ် ပွင့်နေသော ဖိုင်အားလုံးကို print ထုတ်ပေးသော `opensnoop` ကဲ့သို့သော **အသုံးဝင်သော tool အများအပြား** (https://www.brendangregg.com/blog/2015-09-22/bcc-linux-4.3-tracing.html) ပါဝင်သည့် **`bcc`** (https://github.com/iovisor/bcc) ကဲ့သို့သော toolchain တစ်ခုကို အသုံးပြု၍ eBPF ပရိုဂရမ်များကို C ဖြင့် တိုက်ရိုက် ရေးသားနိုင်သည်။

`strace` သည် “ချက်ချင်း စတင်၍ run ရုံသာ” ဖြစ်သဖြင့် အသုံးဝင်သော်လည်း `bpftrace` မှာမူ ပိုမို နည်းပါးသော overhead လိုအပ်သည့်အခါ၊ kernel function များကို စောင့်ကြည့်ဆွဲထုတ်လိုသည့်အခါ သို့မဟုတ် မည်သည့် စုစည်းချက်မျိုးမဆို ပြုလုပ်ရန် လိုအပ်သည့်အခါများတွင် သုံးစွဲရမည့် tool ဖြစ်သည်။ သို့သော် `bpftrace` ကို `root` အဖြစ် run ရမည်ဖြစ်ပြီး သီးခြား process တစ်ခုတည်းကိုသာမက တစ်ခုလုံးသော kernel တစ်ခုလုံးကို စောင့်ကြည့်စစ်ဆေးလေ့ရှိသည်ကို သတိပြုပါ။ သီးခြား ပရိုဂရမ်တစ်ခုကို ပစ်မှတ်ထားရန် command အမည် သို့မဟုတ် PID အလိုက် စစ်ထုတ် (filter) နိုင်သည် -

```
# Filter by command name (prints summary on Ctrl-C)
sudo bpftrace -e 'tracepoint:syscalls:sys_enter_* /comm == "bash"/ { @[probe] =
count(); }'

# Trace a specific command from startup using -c (cpid = child PID)
sudo bpftrace -e 'tracepoint:syscalls:sys_enter_* /pid == cpid/ { @[probe] = co
unt(); }' -c 'ls -la'
```

`-c` flag သည် သတ်မှတ်ထားသော command ကို run ပြီး `cpid` ကို ၎င်း၏ PID အဖြစ် သတ်မှတ်ပေးသည်၊ ၎င်းသည် ပရိုဂရမ် စတင်သည့် အချိန်မှစ၍ စောင့်ကြည့်ဆွဲထုတ်ရန်အတွက် အသုံးဝင်သည်။

စောင့်ကြည့်ထားသော command ထွက်သွားသောအခါ bpftrace သည် စုစည်းထားသော ရလဒ်များကို print ထုတ်ပေးသည်။

Network Debugging

ကွန်ရက် ပြဿနာများအတွက် `tcpdump` (https://www.man7.org/linux/man-pages/man1/tcpdump.1.html) နှင့် Wireshark (https://www.wireshark.org/) တို့သည် ကွန်ရက် packet များကို ဖမ်းယူ၍ ခါတ်ခွဲစစ်ဆေးနိုင်စေသည် -

```
# Capture packets on port 80
sudo tcpdump -i any port 80

# Capture and save to file for Wireshark analysis
sudo tcpdump -i any -w capture.pcap
```

HTTPS traffic များအတွက်မူ encryption ကြောင့် tcpdump သည် အသုံးဝင်မှု နည်းပါးသွားသည်။ mitmproxy (https://mitmproxy.org/) ကဲ့သို့သော tool များသည် encrypt လုပ်ထားသော traffic များကို စစ်ဆေးရန် intercepting proxy တစ်ခုအဖြစ် ဆောင်ရွက်နိုင်သည်။ Browser developer tool များ (Network tab) သည် web application များမှ HTTPS request များကို debug လုပ်ရန်အတွက် အလွယ်ကူဆုံး နည်းလမ်း ဖြစ်လေ့ရှိသည် — ၎င်းတို့သည် decrypt လုပ်ထားသော request/response ဒေတာများ၊ header များနှင့် ကြာမြင့်ချိန် (timing) များကို ဖော်ပြပေးသည်။

Memory Debugging

Memory bug များ — buffer overflow များ၊ use-after-free၊ memory leak များ — သည် အန္တရာယ် အများဆုံးနှင့် debug လုပ်ရန် အခက်ခဲဆုံး အရာများထဲတွင် ပါဝင်သည်။ ၎င်းတို့သည် ချက်ချင်း crash ဖြစ်လေ့မရှိဘဲ နောက်ပိုင်းတွင်မှ ပြဿနာများ ဖြစ်ပေါ်စေသည့် နည်းလမ်းများဖြင့် memory ကို ပျက်စီးစေလေ့ရှိသည်။

Sanitizer များ

Memory bug များကို ရှာဖွေနိုင်သည့် နည်းလမ်းတစ်ခုမှာ runtime ၌ အမှားများကို စမ်းသပ်ရှာဖွေနိုင်ရန် သင့်ကုဒ်တွင် စစ်ဆေးချက်များ ထည့်သွင်းပေးသည့် compiler ၏ feature များ ဖြစ်ကြသော **Sanitizer များ** ကို အသုံးပြုခြင်း ဖြစ်သည်။ ဥပမာအားဖြင့် ကျယ်ပြန့်စွာ အသုံးပြုကြသော **AddressSanitizer (ASan)** သည် အောက်ပါတို့ကို ရှာဖွေပေးသည် - - Buffer overflow များ (stack၊ heap နှင့် global) - Use-after-free - Use-after-return - Memory leak များ

```
# Compile with AddressSanitizer
gcc -fsanitize=address -g program.c -o program
./program
```

အသုံးဝင်သော sanitizer အမျိုးမျိုး ရှိကြသည် -

- **ThreadSanitizer (TSan)**- Multithreaded ကုဒ်များတွင် data race များကို ရှာဖွေပေးသည် (`-fsanitize=thread`)
- **MemorySanitizer (MSan)**- Initialized မလုပ်ရသေးသော memory များကို ဖတ်ရှုခြင်းကို ရှာဖွေပေးသည် (`-fsanitize=memory`)
- **UndefinedBehaviorSanitizer (UBSan)**- Integer overflow ကဲ့သို့သော undefined behavior များကို ရှာဖွေပေးသည် (`-fsanitize=undefined`)

Sanitizer များသည် ပြန်လည် compile လုပ်ရန် လိုအပ်သော်လည်း CI pipeline များနှင့် ပုံမှန် ရေးသား ပြင်ဆင်ချိန်များတွင် အသုံးပြုနိုင်လောက်အောင် မြန်ဆန်ကြသည်။

Valgrind- ပြန်လည် compile မလုပ်နိုင်သည့်အခါ

Valgrind (https://valgrind.org/) မှာမူ memory အမှားများကို ရှာဖွေရန် သင့်ပရိုဂရမ်ကို virtual machine ကဲ့သို့ သော အရာတစ်ခုတွင် run ပေးသည်။ ၎င်းသည် sanitizer များထက် ပိုမိုနှေးကွေးသော်လည်း ပြန်လည် compile လုပ်ရန် မလိုပါ -

```
valgrind --leak-check=full ./my_program
```

အောက်ပါ အခြေအနေများတွင် Valgrind ကို အသုံးပြုပါ - - သင့်ထံတွင် source code မရှိသည့်အခါ - ပြန်လည် compile မလုပ်နိုင်သည့်အခါ (third-party library များ) - Sanitizer အဖြစ် မရရှိနိုင်သော သီးခြား tool များကို လိုအပ်သည့်အခါ

Valgrind သည် အမှန်တကယ်ပင် စွမ်းအားထက်မြက်သည့် ထိန်းချုပ်ထားသော လုပ်ဆောင်ချက် ပတ်ဝန်းကျင်တစ်ခု ဖြစ်ပြီး Profiling အပိုင်းသို့ ရောက်ရှိသည့်အခါ ၎င်းအကြောင်းကို ထပ်မံတွေ့ရှိရမည် ဖြစ်သည်!

Debugging အတွက် AI

Large language model (LLM) များသည် အံ့အားသင့်ဖွယ် အသုံးဝင်သော debugging အကူအညီပေးသူများ ဖြစ်လာကြသည်။ ၎င်းတို့သည် ထုံးတမ်းစဉ်လာ tool များကို အထောက်အကူပြုနိုင်သည့် သီးခြား debugging လုပ်ဆောင်ချက်များတွင် အထူးပင် ကျွမ်းကျင်ကြသည်။

LLM များ ပိုမို အသုံးဝင်သော နေရာများ -

- **နားလည်ရခက်သော Error Message များကို ရှင်းပြခြင်း**- Compiler error များ၊ အထူးသဖြင့် C++ template များ သို့မဟုတ် Rust ၏ borrow checker မှ လာသော error များသည် နားလည်ရခက်ခဲလှသည်။ LLM များသည် ၎င်းတို့အား ရိုးရှင်းသော အင်္ဂလိပ်ဘာသာစကားဖြင့် ပြန်ဆိုပေးနိုင်ပြီး ပြင်ဆင်ချက်များကို အကြံပြုပေးနိုင်သည်။

- **ဘာသာစကားနှင့် Abstraction နယ်နိမိတ်များကို ကျော်လွှန် စေခိုင်းခြင်း**- အကယ်၍ သင်သည် ဘာသာစကား အများအပြား ပါဝင်နေသော ပြဿနာတစ်ခုကို debug လုပ်နေရပါက (ဥပမာ- Python binding မှတစ်ဆင့် ပေါ်ပေါက်လာသော C library တစ်ခုအတွင်းရှိ bug မျိုး) LLM များသည် မတူညီသော အလွှာများကို ချိတ်ဆက် စစ်ဆေးပေးနိုင်သည်။ ၎င်းတို့သည် FFI boundary များ၊ build system ပြဿနာများနှင့် ဘာသာစကားချင်း ချိတ်ဆက်ထားသော debugging များကို နားလည်ရာတွင် အထူးပင် ကောင်းမွန်ကြသည် (ဥပမာ- “ကျွန်ုပ်၏ ပရိုဂရမ်တွင် error တက်နေသည်၊ သို့သော် ၎င်းမှာ ကျွန်ုပ် သုံးထားသော dependency တစ်ခု၏ bug ကြောင့်ဖြစ်သည်ဟု ထင်ပါသည်”)။

- **ရောဂါလက္ခဏာများနှင့် အဓိက အကြောင်းအရင်းများကို ချိတ်ဆက်စစ်ဆေးခြင်း**- “ကျွန်ုပ်၏ ပရိုဂရမ်သည် ပုံမှန် အလုပ်လုပ်သော်လည်း မျှော်မှန်းထားသည်ထက် memory ၁၀ ဆမက ပိုမိုသုံးစွဲနေသည်” ဆိုသည်မှာ LLM များ စုံစမ်းစစ်ဆေးကူညီပေးနိုင်သော ဝေဝါးသည့် လက္ခဏာမျိုး ဖြစ်ပြီး ဖြစ်နိုင်ဖွယ် အကြောင်းအရင်းများနှင့် မည်သည်ကို ရှာဖွေရမည်ကို အကြံပြုပေးနိုင်သည်။

- **Crash dump များနှင့် Stack trace များကို ဓါတ်ခွဲစစ်ဆေးခြင်း**- Stack trace တစ်ခုကို paste လုပ်၍ မည်သည့်အရာကြောင့် ဖြစ်နိုင်သည်ကို မေးမြန်းနိုင်သည်။

Debug symbol များအကြောင်း မှတ်ချက်- အဓိပ္ပာယ်ရှိသော stack trace များနှင့် debugging များအတွက် သင့် binary များ (နှင့် ချိတ်ဆက်ထားသော library မည်သည့်အရာမဆို) ကို debug symbol များ (`-g` flag) ဖြင့် compile လုပ်ထားကြောင်း သေချာပါစေ။ Debug အချက်အလက်များကို ပုံမှန်အားဖြင့် DWARF format ဖြင့် သိမ်းဆည်းလေ့ရှိသည်။ ထို့အပြင် frame pointer များ (`-fno-omit-frame-pointer`) ဖြင့် compile ပြုလုပ်ခြင်းသည် အထူးသဖြင့် profiling tool များအတွက် stack trace များကို ပိုမို စိတ်ချယုံကြည်ရစေသည်။ ၎င်းတို့ မပါရှိပါက stack trace များသည် memory address များကိုသာ ပြသနိုင်သည် သို့မဟုတ် မပြည့်စုံဘဲ ဖြစ်နေနိုင်သည်။ ၎င်းသည် Python သို့မဟုတ် Java ထက် Native compile ပြုလုပ်ထားသော ပရိုဂရမ်များ (C++, Rust) အတွက် ပိုမို အရေးပါသည်။

သတိပြုရမည့် အကန့်အသတ်များ - - LLM များသည် ဖြစ်နိုင်ဖွယ်ရှိပုံရသော်လည်း မှားယွင်းသော ရှင်းလင်းချက်များကို hallucinate လုပ်၍ ပေးတတ်သည် - ၎င်းတို့သည် bug ကို အမှန်တကယ် ပြင်ဆင်ပေးခြင်းထက် ဖုံးကွယ်သွားစေမည့် ပြင်ဆင်ချက်များကို အကြံပြုနိုင်သည် - အကြံပြုချက်များကို အမှန်တကယ် debugging tool များဖြင့် အမြဲမပြတ် စစ်ဆေးပါ - ၎င်းတို့သည် သင့်ကုဒ်အား နားလည်မှုကို အစားထိုးရန် မဟုတ်ဘဲ ဖြည့်စွက်ကူညီပေးသူအဖြစ် သုံးစွဲသည့်အခါ အကောင်းဆုံး အလုပ်လုပ်သည်

၎င်းသည် Development Environment ပို့ချချက်တွင် ဆွေးနွေးခဲ့သည့် [ယေဘုယျ AI ကုဒ်ရေးသားနိုင်စွမ်းများ](https://missing-semester-my.github.io/2026/development-environment/#ai-powered-development) နှင့် ကွဲပြားပါသည်။ ဤနေရာတွင် ကျွန်ုပ်တို့သည် LLM များကို debugging အထောက်အကူအဖြစ် သီးခြား အသုံးပြုခြင်းအကြောင်းကို ပြောဆိုနေခြင်း ဖြစ်သည်။

Profiling

သင့်ကုဒ်သည် လုပ်ဆောင်ချက်ဆိုင်ရာ အရ သင်မျှော်မှန်းထားသည့်အတိုင်း အလုပ်လုပ်နေသည့်တိုင် လုပ်ဆောင်နေစဉ်အတွင်း သင့် CPU သို့မဟုတ် memory အားလုံးကို ကုန်ဆုံးစေပါက ထိုသို့ဖြစ်ခြင်းမှာ ကောင်းမွန်လှမည် မဟုတ်ပေ။ Algorithm သင်ခန်းစာများတွင် Big *O* notation ကို သင်ကြားပေးလေ့ရှိသော်လည်း သင့်ပရိုဂရမ်များရှိ အလုပ်အများဆုံးလုပ်ရသော Hot spot များကို မည်သို့ရှာဖွေရမည်ကို သင်ကြားမပေးကြပါ။ [အချိန်မတန်မီ အလျင်စလို Optimization ပြုလုပ်ခြင်းသည် ဒုက္ခအားလုံး၏ အမြစ်တွယ်ရာဖြစ်သောကြောင့်](https://wiki.c2.com/?PrematureOptimization) Profiler များနှင့် Monitoring tool များကို လေ့လာသင့်သည်။ ၎င်းတို့သည် သင့်ပရိုဂရမ်၏ မည်သည့်အပိုင်းများက အချိန်နှင့်/သို့မဟုတ် အရင်းအမြစ်အများဆုံး သုံးစွဲနေသည်ကို နားလည်စေရန် ကူညီပေးမည်ဖြစ်ပြီး ထိုအပိုင်းများကို ဇောက်ထိုးစိုက်၍ အကောင်းဆုံးဖြစ်အောင် ပြင်ဆင် (optimize) နိုင်မည် ဖြစ်သည်။

Timing (အချိန်တိုင်းတာခြင်း)

စွမ်းဆောင်ရည်ကို တိုင်းတာရန် အလွယ်ကူဆုံး နည်းလမ်းမှာ ကြာမြင့်ချိန်ကို တိုင်းတာခြင်း ဖြစ်သည်။ အခြေအနေ အများအပြားတွင် သင့်ကုဒ်၏ အမှတ်နှစ်ခုကြား ကြာမြင့်ခဲ့သော အချိန်ကို print ထုတ်လိုက်ရုံဖြင့် လုံလောက်နိုင်သည်။

သို့သော်လည်း သင့်ကွန်ပျူတာသည် အခြား process များကို တစ်ပြိုင်နက်တည်း run နေနိုင်သကဲ့သို့ ဖြစ်စဉ်များ ဖြစ်ပျက်လာသည်ကို စောင့်ဆိုင်းနေနိုင်သောကြောင့် Wall clock time (နာရီကြည့်၍ တိုင်းသောကြာချိန်) သည် လွဲမှားစေနိုင်သည်။ `time` command သည် *Real*၊ *User* နှင့် *Sys* အချိန်များအကြား ခွဲခြားပေးသည် -

- **Real** - စတင်ချိန်မှ ပြီးဆုံးချိန်အထိ နာရီကြည့်၍ တိုင်းတာသော ကြာချိန် (စောင့်ဆိုင်းချိန် အပါအဝင်)
- **User** - User ကုဒ်များ run ရန် CPU အတွင်း ကုန်လွန်ခဲ့သော အချိန်
- **Sys** - Kernel ကုဒ်များ run ရန် CPU အတွင်း ကုန်လွန်ခဲ့သော အချိန်

```
$ time curl https://missing.csail.mit.edu &> /dev/null
real    0m0.272s
user    0m0.079s
sys     0m0.028s
```

ဤနေရာတွင် request သည် မီလီစက္ကန့် ၃၀၀ နီးပါး (real time) ကြာမြင့်ခဲ့သော်လည်း CPU အချိန် (user + sys) မှာ ၁၀၇ မီလီစက္ကန့်သာ ရှိခဲ့သည်။ ကျန်ရှိသော အချိန်မှာ ကွန်ရက်ကို စောင့်ဆိုင်းနေခဲ့ခြင်း ဖြစ်သည်။

Resource Monitoring (အရင်းအမြစ် စောင့်ကြည့်စစ်ဆေးခြင်း)

အချို့အခါများတွင် သင့်ပရိုဂရမ်၏ စွမ်းဆောင်ရည်ကို ခါတ်ခွဲစစ်ဆေးရန် ပထမဆုံး အဆင့်မှာ ၎င်း၏ အမှန်တကယ် အရင်းအမြစ် သုံးစွဲမှုကို နားလည်ခြင်း ဖြစ်သည်။ ပရိုဂရမ်များသည် အရင်းအမြစ် ကန့်သတ်ချက်များ ရှိနေသည့်အခါ မကြာခဏ နှေးကွေးစွာ အလုပ်လုပ်လေ့ရှိသည်။

- **ယေဘုယျ စောင့်ကြည့်စစ်ဆေးခြင်း**- `htop` (https://htop.dev/) သည် လက်ရှိ run နေသော process များ၏ စာရင်းအင်း အမျိုးမျိုးကို ဖော်ပြပေးသည့် `top` ၏ မြှင့်တင်ထားသော ဗားရှင်း ဖြစ်သည်။ အသုံးဝင်သော shortcut များ- process များကို စီရန် `<F6>` ၊ သစ်ပင်ပုံစံ အဆင့်ဆင့်ပြရန် `t` ၊ thread များကို အဖွင့်အပိတ်လုပ်ရန် `h` ။ ထို့အပြင် အရာအများအပြားကို *ပို*မို စောင့်ကြည့်ပေးသည့် `btop` (https://github.com/aristocratos/btop) လည်း ရှိသည်။
- **I/O လုပ်ဆောင်ချက်များ**- `iotop` (https://www.man7.org/linux/man-pages/man8/iotop.8.html) သည် တိုက်ရိုက် I/O သုံးစွဲမှု အချက်အလက်များကို ဖော်ပြပေးသည်။
- **Memory သုံးစွဲမှု**- `free` (https://www.man7.org/linux/man-pages/man1/free.1.html) သည် စုစုပေါင်း လွတ်လပ်နေသော Memory နှင့် သုံးစွဲထားသော Memory ကို ဖော်ပြပေးသည်။
- **ပွင့်နေသော ဖိုင်များ**- `lsof` (https://www.man7.org/linux/man-pages/man8/lsof.8.html) သည် process များက ဖွင့်လှစ်ထားသော ဖိုင်များ၏ အချက်အလက်များကို ဖော်ပြပေးသည်။ သီးခြားဖိုင်တစ်ခုကို မည်သည့် process က ဖွင့်ထားသည်ကို စစ်ဆေးရာတွင် အသုံးဝင်သည်။
- **ကွန်ရက် ချိတ်ဆက်မှုများ**- `ss` (https://www.man7.org/linux/man-pages/man8/ss.8.html) သည် ကွန်ရက် ချိတ်ဆက်မှုများကို စောင့်ကြည့်နိုင်စေသည်။ အတွေ့ရများသော အသုံးပြုပုံမှာ မည်သည့် process က ပေးထားသော port ကို အသုံးပြုနေသည်ကို ရှာဖွေခြင်းဖြစ်သည်- `ss -tlnp | grep :8080` ။
- **ကွန်ရက် သုံးစွဲမှု**- `nethogs` (https://github.com/raboof/nethogs) နှင့် `iftop` (https://pdw.ex-parrot.com/iftop/) တို့သည် process တစ်ခုချင်းစီအလိုက် ကွန်ရက် သုံးစွဲမှုကို စောင့်ကြည့်ရန် ကောင်းမွန်သော Interactive CLI tool များ ဖြစ်ကြသည်။

Performance Data များကို ပုံဖော် ကြည့်ရှုခြင်း

လူများသည် ကိန်းဂဏာန်း ဇယားများထက် graph များကွေးများတွင် ပုံစံ (pattern) များကို ပိုမို မြန်ဆန်စွာ သတိပြုမိကြသည်။ စွမ်းဆောင်ရည်ကို ခါတ်ခွဲစစ်ဆေးသည့်အခါ သင့်ဒေတာများကို graph ဆွဲကြည့်ခြင်းဖြင့် ကိန်းဂဏာန်း အကြမ်းများတွင် မမြင်နိုင်သော Trend များ၊ Spike များ နှင့် မူမမှန်မှုများကို ဖော်ထုတ်ပေးလေ့ ရှိသည်။

ဒေတာများကို graph ဆွဲရလွယ်ကူအောင် ပြုလုပ်ခြင်း- Debugging အတွက် print သို့မဟုတ် log statement များ ထည့်သွင်းသည့်အခါ နောက်ပိုင်းတွင် လွယ်ကူစွာ graph ဆွဲနိုင်ရန် output ကို format လုပ်ရန် စဉ်းစားပါ။

CSV format ဖြင့် သာမန် timestamp နှင့် တန်ဖိုး (`1705012345,42.5`) သည် စကားပြေ စာကြောင်းထက် graph ဆွဲရသည်မှာ ပိုမို လွယ်ကူသည်။ JSON-structured log များကိုလည်း နည်းပါးသော အားထုတ်မှုဖြင့် parse လုပ်၍ graph ဆွဲနိုင်သည်။ အခြားစကားဖြင့် ပြောရလျှင် သင့်ဒေတာများကို [သပ်သပ်ရပ်ရပ် ရေးသားပါ](https://vita.had.co.nz/papers/tidy-data.pdf) (https://vita.had.co.nz/papers/tidy-data.pdf)။

gnuplot ဖြင့် လျှင်မြန်စွာ graph ဆွဲခြင်း- ရိုးရှင်းသော command-line graph ဆွဲခြင်းအတွက် **`gnuplot`** (http://www.gnuplot.info/) သည် ဒေတာဖိုင်များမှ graph များကို တိုက်ရိုက် ထုတ်လုပ်ပေးနိုင်သည် -

```
# Plot a simple CSV with timestamp,value
gnuplot -e "set datafile separator ','; plot 'latency.csv' using 1:2 with line
s"
```

matplotlib နှင့် ggplot2 ဖြင့် အကြိမ်ကြိမ် ဓါတ်ခွဲ စမ်းသပ်ခြင်း- ပိုမို နက်ရှိုင်းသော ဓါတ်ခွဲစစ်ဆေးမှု အတွက် Python ၏ **`matplotlib`** (https://matplotlib.org/) နှင့် R ၏ **`ggplot2`** (https://ggplot2.tidyverse.org/) တို့သည် အကြိမ်ကြိမ် ဓါတ်ခွဲစမ်းသပ်မှုများကို ပြုလုပ်နိုင်စေသည်။ အခိုက်အတန့် graph ဆွဲခြင်းများနှင့် မတူဘဲ ဤ tool များသည် အယူအဆ (hypothesis) များကို စုံစမ်းစစ်ဆေးရန် ဒေတာများကို မြန်ဆန်စွာ အစိတ်အပိုင်းပိုင်းခြား၍ ပြောင်းလဲနိုင်စေသည်။ ggplot2 ၏ facet plot များသည် အထူးပင် စွမ်းအား ထက်မြက်သည် — အမျိုးအစားအလိုက် (ဥပမာ- request latency ကို endpoint သို့မဟုတ် အချိန်အလိုက် ပိုင်းခြား၍) ဒေတာအစုအဝေး တစ်ခုတည်းကို subplot အများအပြားသို့ ခွဲထုတ်နိုင်ပြီး မဟုတ်ပါက ဖုံးကွယ်နေမည့် ပုံစံများကို ဖော်ထုတ်ပေးနိုင်သည်။

နမူနာ အသုံးပြုမှုများ - - အချိန်အလိုက် request latency ကို graph ဆွဲခြင်းသည် အကြမ်း percentile များက ဖုံးကွယ်ထားသော ပုံမှန် နှေးကွေးမှုများ (garbage collection၊ cron job များ၊ traffic ပုံစံများ) ကို ဖော်ထုတ်ပေးသည် - ကြီးထွားလာသော ဒေတာစထရက်ချာတစ်ခုအတွက် ထည့်သွင်းချိန် (insert time) များကို ပုံဖော်ကြည့်ရှုခြင်းသည် algorithm ၏ ရှုပ်ထွေးမှု ပြဿနာများကို ဖော်ပြပေးနိုင်သည် — vector ထည့်သွင်းမှုများ၏ graph သည် အောက်ခံ array အရွယ်အစား နှစ်ဆဖြစ်သွားသည့်အခါ ထူးခြားသော spike များကို ဖော်ပြမည်ဖြစ်သည် - မတူညီသော Dimension များ (request အမျိုးအစား၊ user အစုအဝေး၊ server) အလိုက် metric များကို ပိုင်းခြားခြင်းသည် “တစ်စနစ်လုံးဆိုင်ရာ” ပြဿနာတစ်ခုသည် အမှန်တကယ်တွင် အမျိုးအစားတစ်ခုတည်း၌သာ သီးခြားဖြစ်ပေါ်နေခြင်းဖြစ်ကြောင်း မကြာခဏ ဖော်ထုတ်ပေးသည်

CPU Profiler များ

လူများက *profiler* အကြောင်း ပြောဆိုသည့်အခါ အချိန်အများစုတွင် *CPU profiler* များကို ရည်ရွယ်ကြခြင်းဖြစ်သည်။ အဓိက အမျိုးအစား နှစ်ခု ရှိသည် -

- **Tracing profiler များ** သည် သင့်ပရိုဂရမ် ပြုလုပ်သော function call တိုင်းကို မှတ်တမ်းတင်ထားကြသည်
- **Sampling profiler များ** သည် သင့်ပရိုဂရမ်ကို ပုံမှန် အချိန်အပိုင်းအခြားအလိုက် (ယေဘုယျအားဖြင့် ၁ မီလီစက္ကန့်တိုင်း) စစ်ဆေးပြီး ပရိုဂရမ်၏ stack ကို မှတ်တမ်းတင်ကြသည်

Sampling profiler များသည် နည်းပါးသော overhead ရှိကြပြီး Production တွင် အသုံးပြုရန် ယေဘုယျအားဖြင့် ပိုမိုနှစ်သက်ကြသည်။

perf- Sampling Profiler

`perf` (https://www.man7.org/linux/man-pages/man1/perf.1.html) သည် စံ Linux profiler ဖြစ်သည်။ ၎င်းသည် ပြန်လည် compile လုပ်ရန် မလိုဘဲ မည်သည့် ပရိုဂရမ်ကိုမဆို profile လုပ်နိုင်သည်။

`perf stat` သည် အချိန် မည်သည့်နေရာတွင် ကုန်လွန်သွားသည်ကို မြန်ဆန်သော သုံးသပ်ချက် ပေးသည် -

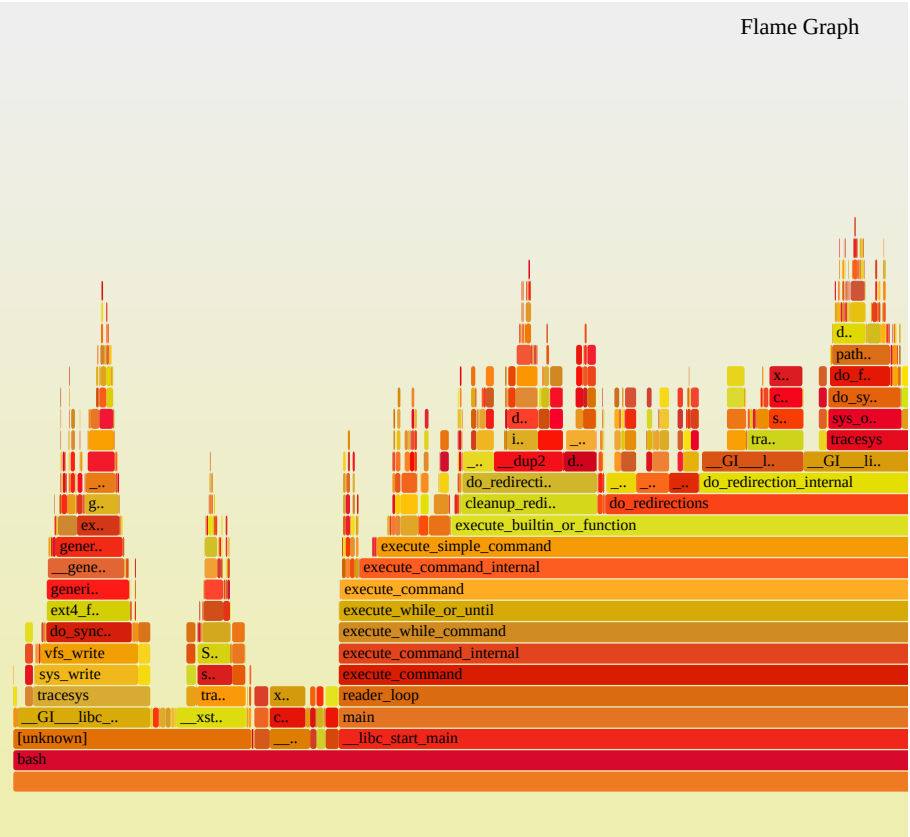
```
$ perf stat ./slow_program

 Performance counter stats for './slow_program':

         3,210.45 msec task-clock                #    0.998 CPUs utilized
               12      context-switches          #    3.738 /sec
                0      cpu-migrations            #    0.000 /sec
              156      page-faults               #   48.587 /sec
   12,345,678,901      cycles                    #    3.845 GHz
    9,876,543,210      instructions              #    0.80  insn per cycle
    1,234,567,890      branches                  #  384.532 M/sec
       12,345,678      branch-misses             #    1.00% of all branches
```

လက်တွေ့ကမ္ဘာ ပရိုဂရမ်များအတွက် Profiler output များသည် ကြီးမားသော အချက်အလက် ပမာဏ ပါဝင်လိမ့်မည်။ လူများသည် အမြင်အာရုံကို ပိုမို အသုံးပြုသော သတ္တဝါများ ဖြစ်ကြပြီး ကြီးမားသော ကိန်းဂဏန်း ပမာဏများကို ဖတ်ရှုရာတွင် တော်တော်ပင် ညံ့ဖျင်းကြသည်။ **Flame graph များ** (https://www.brendangregg.com/flamegraphs.html) သည် profiling ဒေတာများကို ပိုမို နားလည်ရလွယ်ကူစေသည့် ရုပ်ပုံလွှာ ပုံဖော်ခြင်း (visualization) ဖြစ်သည်။

Flame graph တစ်ခုသည် Y axis တွင် function call များ၏ အဆင့်ဆင့် အစဉ်လိုက်ကို ဖော်ပြပြီး X axis တွင် ကုန်လွန်ခဲ့သော အချိန်နှင့် အချိုးကျ ဖော်ပြပေးသည်။ ၎င်းတို့သည် Interactive ဖြစ်ကြသည် — ပရိုဂရမ်၏ သီးခြား အစိတ်အပိုင်းများသို့ ချဲ့ကြည့် (zoom in) ရန် click နှိပ်နိုင်သည်။



`perf` ဒေတာမှ flame graph တစ်ခု ထုတ်လုပ်ရန် -

```
# Record profile
perf record -g ./my_program

# Generate flame graph (requires flamegraph scripts)
perf script | stackcollapse-perf.pl | flamegraph.pl > flamegraph.svg
```

> Interactive web-based flame graph viewer အတွက် **Speedscope** (https://www.speedscope.app/) ကို အသုံးပြုရန် သို့မဟုတ် စနစ်အဆင့် ဘက်စုံ ဓါတ်ခွဲစစ်ဆေးမှုအတွက် **Perfetto** (https://perfetto.dev/) ကို အသုံးပြုရန် စဉ်းစားပါ။

Valgrind ၏ Callgrind- Tracing Profiler

`callgrind` (https://valgrind.org/docs/manual/cl-manual.html) သည် သင့်ပရိုဂရမ်၏ call ရာဇဝင်နှင့် instruction အရေအတွက်များကို မှတ်တမ်းတင်သည့် profiling tool တစ်ခု ဖြစ်သည်။ Sampling profiler များနှင့် မတူဘဲ ၎င်းသည် အတိအကျ call အရေအတွက်များကို ပံ့ပိုးပေးပြီး caller များနှင့် callee များအကြား ဆက်နွယ်မှုကို ဖော်ပြပေးနိုင်သည် -

```
# Run with callgrind
valgrind --tool=callgrind ./my_program

# Analyze with callgrind_annotate (text) or kcachegrind (GUI)
callgrind_annotate callgrind.out.<pid>
kcachegrind callgrind.out.<pid>
```

Callgrind သည် sampling profiler များထက် ပိုမိုနှေးကွေးသော်လည်း တိကျသော call အရေအတွက်များကို ပံ့ပိုးပေးပြီး အကယ်၍ သင်လိုအပ်ပါက cache ပြုမူဆောင်ရွက်ချက်များကိုပါ (`--cache-sim=yes` ဖြင့်) တုပပေးနိုင်သည်။

> အကယ်၍ သင်သည် သီးခြား ဘာသာစကားတစ်ခုကို အသုံးပြုနေပါက ပိုမိုအထူးပြုထားသော profiler များ ရှိနိုင်ပေသည်။ ဥပမာအားဖြင့် Python ၌ `cProfile` (https://docs.python.org/3/library/profile.html) နှင့် `py-spy` (https://github.com/benfred/py-spy) ရှိပြီး၊ Go ၌ `go tool pprof` (https://pkg.go.dev/cmd/pprof) ရှိကာ၊ Rust ၌ `cargo-flamegraph` (https://github.com/flamegraph-rs/flamegraph) (၎င်းမှာ အမှန်တကယ်တွင် compiled ပရိုဂရမ် မည်သည့် အရာအတွက်မဆို အလုပ်လုပ်သည်!) ရှိကြသည်။

Memory Profiler များ

Memory profilerများသည် သင့်ပရိုဂရမ်က အချိန်နှင့်အမျှ Memory မည်သို့ သုံးစွဲနေသည်ကို နားလည်စေရန်နှင့် Memory leak များကို ရှာဖွေနိုင်ရန် ကူညီပေးသည်။

Valgrind ၏ Massif

`massif` (https://valgrind.org/docs/manual/ms-manual.html) သည် Heap memory သုံးစွဲမှုကို profile လုပ်ပေးသည် -

```
valgrind --tool=massif ./my_program
ms_print massif.out.<pid>
```

၎င်းသည် အချိန်နှင့်အမျှ heap သုံးစွဲမှုကို ဖော်ပြပေးပြီး memory leak များနှင့် အလွန်အမင်း သတ်မှတ်ထားမှု (excessive allocation) များကို ခွဲခြားသိရှိနိုင်ရန် ကူညီပေးသည်။

> Python အတွက်မူ `memory-profiler` (https://pypi.org/project/memory-profiler/) သည် တစ်လိုင်းချင်းစီ အလိုက် memory သုံးစွဲမှု အချက်အလက်ကို ပံ့ပိုးပေးသည်။

Benchmarking (စွမ်းဆောင်ရည် နှိုင်းယှဉ်တိုင်းတာခြင်း)

မတူညီသော ရေးသားမှုများ သို့မဟုတ် tool များ၏ စွမ်းဆောင်ရည်ကို နှိုင်းယှဉ်ရန် လိုအပ်သည့်အခါ `hyperfine` (https://github.com/sharkdp/hyperfine) သည် command-line ပရိုဂရမ်များကို benchmark ပြုလုပ်ရန် အလွန်ကောင်းမွန်သည် -

```
$ hyperfine --warmup 3 'fd -e jpg' 'find . -iname "*.jpg"'
Benchmark #1: fd -e jpg
  Time (mean ± σ):      51.4 ms ±   2.9 ms    [User: 121.0 ms, System: 160.5 m
s]
  Range (min … max):    44.2 ms …  60.1 ms    56 runs

Benchmark #2: find . -iname "*.jpg"
  Time (mean ± σ):      1.126 s ±  0.101 s    [User: 141.1 ms, System: 956.1 m
s]
  Range (min … max):    0.975 s …  1.287 s    10 runs

Summary
  'fd -e jpg' ran
   21.89 ± 2.33 times faster than 'find . -iname "*.jpg"'
```

> Web development အတွက်မူ browser developer tool များတွင် ကောင်းမွန်လှသော profiler များ ပါဝင်သည်။ **Firefox Profiler** (https://profiler.firefox.com/docs/) နှင့် **Chrome DevTools** (https://developers.google.com/web/tools/chrome-devtools/rendering-tools) မှတ်တမ်းများကို ကြည့်ရှုပါ။

လေ့ကျင့်ခန်းများ

Debugging

1. **Sort ပြုလုပ်သော Algorithm တစ်ခုကို Debug လုပ်ပါ**- အောက်ပါ pseudocode သည် merge sort ကို ရေးသားထားခြင်း ဖြစ်သော်လည်း bug တစ်ခု ပါဝင်နေသည်။ ၎င်းအား သင်နှစ်သက်ရာ ဘာသာစကား တစ်ခုဖြင့် ရေးသားပါ၊ ထို့နောက် bug ကို ရှာဖွေပြင်ဆင်ရန် debugger (gdb၊ lldb၊ pdb သို့မဟုတ် သင့် IDE ၏ debugger) တစ်ခုကို အသုံးပြုပါ။

```
function merge_sort(arr):
    if length(arr) <= 1:
        return arr
    mid = length(arr) / 2
    left = merge_sort(arr[0..mid])
    right = merge_sort(arr[mid..end])
    return merge(left, right)

function merge(left, right):
    result = []
    i = 0, j = 0
    while i < length(left) AND j < length(right):
        if left[i] <= right[j]:
            append result, left[i]
            i = i + 1
        else:
            append result, right[i]
            j = j + 1
    append remaining elements from left and right
    return result
```

   စမ်းသပ်မှုဒေတာ- `merge_sort([3, 1, 4, 1, 5, 9, 2, 6])` သည် `[1, 1, 2, 3, 4, 5, 6, 9]` ကို ပြန်ပေးရမည် ဖြစ်သည်။ မမှန်ကန်သော element ကို ရွေးချယ်နေသည့် နေရာကို ရှာဖွေရန် breakpoint များကို သုံး၍ merge function အတွင်းသို့ တစ်ဆင့်ချင်းဝင်ရောက် (step through) စစ်ဆေးပါ။

2. `rr` (https://rr-project.org/) ကို install လုပ်ပြီး ပျက်စီးမှု bug (corruption bug) တစ်ခုကို ရှာဖွေရန် reverse debugging ကို အသုံးပြုပါ။ ဤပရိုဂရမ်ကို `corruption.c` အဖြစ် သိမ်းဆည်းပါ -

```
#include <stdio.h>

typedef struct {
    int id;
    int scores[3];
} Student;

Student students[2];

void init() {
    students[0].id = 1001;
    students[0].scores[0] = 85;
    students[0].scores[1] = 92;
    students[0].scores[2] = 78;

    students[1].id = 1002;
    students[1].scores[0] = 90;
    students[1].scores[1] = 88;
    students[1].scores[2] = 95;
}

void curve_scores(int student_idx, int curve) {
    for (int i = 0; i < 4; i++) {
        students[student_idx].scores[i] += curve;
    }
}

int main() {
    init();
    printf("=== Initial state ===\n");
    printf("Student 0: id=%d\n", students[0].id);
    printf("Student 1: id=%d\n", students[1].id);

    curve_scores(0, 5);

    printf("\n=== After curving ===\n");
    printf("Student 0: id=%d\n", students[0].id);
    printf("Student 1: id=%d\n", students[1].id);

    if (students[1].id != 1002) {
        printf("\nERROR: Student 1's ID was corrupted! Expected 1002, got %d
\n",
               students[1].id);
        return 1;
    }
```

```
    return 0;
}
```

`gcc -g corruption.c -o corruption` ဖြင့် compile လုပ်၍ run ပါ။ Student 1 ၏ ID သည် ပျက်စီးသွားသည်၊ သို့သော် ပျက်စီးမှုမှာ student 0 ကိုသာ ကိုင်တွယ်သော function တစ်ခုအတွင်း ဖြစ်ပွားခဲ့ခြင်းဖြစ်သည်။ တရားခံကို ရှာဖွေရန် `rr record ./corruption` နှင့် `rr replay` ကို အသုံးပြုပါ။ `students[1].id` ပေါ်တွင် watchpoint တစ်ခု သတ်မှတ်ပြီး ပျက်စီးသွားပြီးနောက် မည်သည့် ကုဒ်လိုင်းက ၎င်းအား ထပ်မံကျော်ရေးသွားခဲ့သည်ကို အတိအကျ ရှာဖွေရန် `reverse-continue` ကို အသုံးပြုပါ။

3. AddressSanitizer ဖြင့် Memory အမှားတစ်ခုကို Debug လုပ်ပါ။ ၎င်းကို `uaf.c` အဖြစ် သိမ်းဆည်းပါ -

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main() {
    char *greeting = malloc(32);
    strcpy(greeting, "Hello, world!");
    printf("%s\n", greeting);

    free(greeting);

    greeting[0] = 'J';
    printf("%s\n", greeting);

    return 0;
}
```

ပထမဦးစွာ sanitizer များ မပါဘဲ compile လုပ်၍ run ပါ- `gcc uaf.c -o uaf && ./uaf` ။ ၎င်းမှာ အလုပ်လုပ်နေပုံ ပေါ်နိုင်သည်။ ယခု AddressSanitizer ဖြင့် compile ပြုလုပ်ပါ- `gcc -fsanitize=address -g uaf.c -o uaf && ./uaf` ။ Error report ကို ဖတ်ရှုပါ။ ASan က မည်သည့် bug ကို ရှာတွေ့သနည်း။ ၎င်းတွေ့ရှိသည့် ပြဿနာကို ပြင်ဆင်ပါ။

4. `ls -l` ကဲ့သို့သော command တစ်ခုက ပြုလုပ်သည့် system call များကို စောင့်ကြည့်ဆွဲထုတ်ရန် `strace` (Linux) သို့မဟုတ် `dtruss` (macOS) ကို အသုံးပြုပါ။ ၎င်းသည် မည်သည့် system call များကို ပြုလုပ်နေသနည်း။ ပိုမို ရှုပ်ထွေးသော ပရိဂရမ်တစ်ခုကို စောင့်ကြည့်ဆွဲထုတ်ကြည့်ပြီး ၎င်းက မည်သည့်ဖိုင်များကို ဖွင့်လှစ်သည်ကို ကြည့်ပါ။

5. နားလည်ရခက်သော error message တစ်ခုကို debug လုပ်ကူရန် LLM ကို အသုံးပြုပါ။ Compiler error တစ်ခုကို (အထူးသဖြင့် C++ template များ သို့မဟုတ် Rust မှ) ကူးယူ၍ ရှင်းလင်းချက်နှင့် ပြင်ဆင်ချက် တောင်းဆိုကြည့်ပါ။ `strace` သို့မဟုတ် address sanitizer မှ output အချို့ကို ၎င်းအတွင်းသို့ ထည့်သွင်းကြည့်ပါ။

Profiling

1. သင်နှစ်သက်ရာ ပရိုဂရမ်တစ်ခုအတွက် အခြေခံ စွမ်းဆောင်ရည် စာရင်းအင်းများကို ရယူရန် `perf stat` ကို အသုံးပြုပါ။ မတူညီသော counter များသည် မည်သည့်အရာကို ဆိုလိုသနည်း။
2. `perf record` ဖြင့် profile ပြုလုပ်ပါ။ ၎င်းကို `slow.c` အဖြစ် သိမ်းဆည်းပါ -

```c
#include <math.h>
#include <stdio.h>

double slow_computation(int n) {
    double result = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < 1000; j++) {
            result += sin(i * j) * cos(i + j);
        }
    }
    return result;
}

int main() {
    double r = 0;
    for (int i = 0; i < 100; i++) {
        r += slow_computation(1000);
    }
    printf("Result: %f\n", r);
    return 0;
}
```

Debug symbol များနှင့်အတူ compile ပြုလုပ်ပါ- `gcc -g -O2 slow.c -o slow -lm` ။ `perf record -g ./slow` ကို run ပါ၊ ထို့နောက် အချိန် မည်သည့်နေရာတွင် ကုန်လွန်သည်ကို ကြည့်ရန် `perf report` ကို run ပါ။ flamegraph script များကို အသုံးပြု၍ flame graph တစ်ခု ထုတ်လုပ်ကြည့်ပါ။

3. အလုပ်တစ်ခုတည်း၏ မတူညီသော ရေးသားချက် နှစ်ခုကို benchmark တိုင်းတာရန် `hyperfine` ကို အသုံးပြုပါ (ဥပမာ- `find` နှင့် `fd` ၊ `grep` နှင့် `ripgrep` သို့မဟုတ် သင့်ကိုယ်ပိုင်ကုဒ်၏ ဗားရှင်းနှစ်

ခု)။

4. အရင်းအမြစ် များစွာသုံးသော ပရိုဂရမ်တစ်ခုကို run နေစဉ် သင့်စနစ်ကို စောင့်ကြည့်ရန် `htop` ကို အသုံးပြုပါ။ process တစ်ခု အသုံးပြုနိုင်သော CPU များကို ကန့်သတ်ရန် `taskset` ကို သုံးကြည့်ပါ- `taskset --cpu-list 0,2 stress -c 3` ။ `stress` သည် အဘယ်ကြောင့် CPU သုံးခုကို မသုံးသနည်း။

5. အတွေ့ရများသော ပြဿနာတစ်ခုမှာ သင်စောင့်နားထောင်လိုသော port ကို အခြား process တစ်ခုက ရယူထားပြီး ဖြစ်နေခြင်း ဖြစ်သည်။ ထို process ကို မည်သို့ရှာဖွေရမည်ကို လေ့လာပါ- ပထမဦးစွာ port 4444 တွင် အနည်းဆုံး web server တစ်ခု စတင်ရန် `python -m http.server 4444` ကို အကောင်အထည်ဖော်ပါ။ သီးခြား terminal တစ်ခုတွင် process ကို ရှာဖွေရန် `ss -tlnp | grep 4444` ကို run ပါ။ ၎င်းအား `kill <PID>` ဖြင့် အဆုံးသတ်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. `gdb`	https://www.gnu.org/software/gdb/	
2. `lldb`	https://lldb.llvm.org/	
3. rr	https://rr-project.org/	
4. Warpspeed	https://warpspeed.dev/	
5. system call များ	https://en.wikipedia.org/wiki/System_call	
6. `strace`	https://www.man7.org/linux/man-pages/man1/strace.1.html	
7. `dtruss`	https://www.manpagez.com/man/1/dtruss/	
8. strace zine	https://jvns.ca/strace-zine-unfolded.pdf	
9. eBPF	https://ebpf.io/	
10. `bpftrace`	https://github.com/iovisor/bpftrace	
11. အသုံးဝင်သော tool အများအပြား	https://www.brendangregg.com/blog/2015-09-22/bcc-linux-4.3-tracing.html	
12. `bcc`	https://github.com/iovisor/bcc	
13. `tcpdump`	https://www.man7.org/linux/man-pages/man1/tcpdump.1.html	
14. Wireshark	https://www.wireshark.org/	
15. mitmproxy	https://mitmproxy.org/	

Resource / Description	URL	Micro QR
16. Valgrind	https://valgrind.org/	
17. ယေဘုယျ AI ကုဒ်ရေးသားနိုင်စွမ်းများ	https://missing-semester-my.github.io/2026/development-environment/#ai-powered-development	
18. အချိန်မတန်မီ အလျင်လို Optimization ပြုလုပ်ခြင်းသည် ဒုက္ခအားလုံး၏ အမြစ်တွယ်ရာဖြစ်သောကြောင့်	https://wiki.c2.com/?PrematureOptimization	
19. `htop`	https://htop.dev/	
20. `btop`	https://github.com/aristocratos/btop	
21. `iotop`	https://www.man7.org/linux/man-pages/man8/iotop.8.html	
22. `free`	https://www.man7.org/linux/man-pages/man1/free.1.html	
23. `lsof`	https://www.man7.org/linux/man-pages/man8/lsof.8.html	
24. `ss`	https://www.man7.org/linux/man-pages/man8/ss.8.html	
25. `nethogs`	https://github.com/raboof/nethogs	
26. `iftop`	https://pdw.ex-parrot.com/iftop/	
27. သပ်သပ်ရပ်ရပ် ရေးသားပါ	https://vita.had.co.nz/papers/tidy-data.pdf	
28. `gnuplot`	http://www.gnuplot.info/	
29. `matplotlib`	https://matplotlib.org/	
30. `ggplot2`	https://ggplot2.tidyverse.org/	
31. `perf`	https://www.man7.org/linux/man-pages/man1/perf.1.html	

Resource / Description	URL	Micro QR
32. Flame graph ပုံ	https://www.brendangregg.com/flamegraphs.html	
33. ![FlameGraph	https://www.brendangregg.com/FlameGraphs/cpu-bash-flamegraph.svg	
34. Speedscope	https://www.speedscope.app/	
35. Perfetto	https://perfetto.dev/	
36. `callgrind`	https://valgrind.org/docs/manual/cl-manual.html	
37. `cProfile`	https://docs.python.org/3/library/profile.html	
38. `py-spy`	https://github.com/benfred/py-spy	
39. `go tool pprof`	https://pkg.go.dev/cmd/pprof	
40. `cargo-flamegraph`	https://github.com/flamegraph-rs/flamegraph	
41. `massif`	https://valgrind.org/docs/manual/ms-manual.html	
42. `memory-profiler`	https://pypi.org/project/memory-profiler/	
43. `hyperfine`	https://github.com/sharkdp/hyperfine	
44. Firefox Profiler	https://profiler.firefox.com/docs/	
45. Chrome DevTools	https://developers.google.com/web/tools/chrome-devtools/rendering-tools	

Development Environment နှင့် Tools များ

အနှစ်ချုပ် - IDEs၊ Vim၊ language servers နှင့် AI စွမ်းအားသုံး ဆော့ဖ်ဝဲလ် ရေးသားရေး tools များ အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Development Environment နှင့် Tools များ

Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=QnM1nVzrkx8

Development environment (ဆော့ဖ်ဝဲလ် ရေးသားသည့် ပတ်ဝန်းကျင်) ဆိုသည်မှာ ဆော့ဖ်ဝဲလ် ရေးသားထုတ်လုပ်ရန်အတွက် အသုံးပြုသည့် tools များ အစုအဝေး ဖြစ်သည်။ Development environment တစ်ခု၏ အဓိက ဗဟိုချက်မှာ စာသား ပြင်ဆင်တည်းဖြတ်သည့် (text editing) လုပ်ဆောင်ချက် ဖြစ်ပြီး၊ ၎င်းနှင့်အတူ syntax highlighting၊ type checking၊ code formatting နှင့် autocomplete ကဲ့သို့သော တွဲဖက် လုပ်ဆောင်ချက်များ ပါဝင်သည်။ **VS Code** ကဲ့သို့သော *Integrated development environments* (IDEs) များသည် ဤလုပ်ဆောင်ချက် အားလုံးကို ဆော့ဖ်ဝဲလ် အက်ပလီကေးရှင်း တစ်ခုတည်းအတွင်း စုစည်း ပေးထားသည်။ Terminal အခြေပြု ဆော့ဖ်ဝဲလ် ရေးသားမှု လုပ်ငန်းစဉ်များ (Terminal-based development workflows) တွင်မူ **tmux** (https://github.com/tmux/tmux) (terminal multiplexer တစ်ခု)၊ **Vim** (https://www.vim.org/) (text editor တစ်ခု)၊ **Zsh** (https://www.zsh.org/) (shell တစ်ခု) နှင့် **Ruff** (https://docs.astral.sh/ruff/) (Python linter နှင့် code formatter တစ်ခု)၊ **Mypy** (https://mypy-lang.org/) (Python type checker တစ်ခု) တို့ကဲ့သို့သော သက်ဆိုင်ရာ ပရိဂရမ်းမင်း ဘာသာစကားအလိုက် သီးသန့် အသုံးပြုသည့် command-line tools များကို ပေါင်းစပ် အသုံးပြုကြသည်။

IDEs နှင့် terminal အခြေပြု လုပ်ငန်းစဉ်များတွင် ၎င်းတို့၏ အားသာချက်၊ အားနည်းချက်များ အသီးသီး ရှိကြသည်။ ဥပမာအားဖြင့် Graphical IDEs များသည် လေ့လာရ ပိုမို လွယ်ကူနိုင်ပြီး ယနေ့ခေတ် IDEs များတွင် AI autocomplete ကဲ့သို့သော AI ပေါင်းစပ် လုပ်ဆောင်ချက်များ အသင့် ပါဝင်လေ့ ရှိသည်။ အခြားတစ်ဖက်တွင်လည်း Terminal အခြေပြု လုပ်ငန်းစဉ်များသည် ပေါ့ပါးပြီး GUI မရှိသော သို့မဟုတ် ဆော့ဖ်ဝဲလ် အသစ် တပ်ဆင်ခွင့် မရှိသော ပတ်ဝန်းကျင်များတွင် အသုံးပြုနိုင်သည့် တစ်ခုတည်းသော နည်းလမ်း ဖြစ်နိုင်သည်။ သင့်အနေဖြင့် နည်းလမ်း နှစ်ခုလုံးနှင့် အခြေခံကျွမ်းကျင်မှု ရှိထားပြီး အနည်းဆုံး နည်းလမ်း တစ်ခုကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်ရန် အကြံပြုလိုပါသည်။ သင့်တွင် အသုံးပြုနေကျ IDE မရှိသေးပါက **VS Code** ဖြင့် စတင်ရန် အကြံပြုပါသည်။

ဒီသင်ခန်းစာမှာ ကျွန်တော်တို့ အောက်ပါ အကြောင်းအရာများကို ဆွေးနွေးသွားပါမည်-

- **စာသား တည်းဖြတ်ခြင်း နှင့် Vim**
- **Code intelligence နှင့် language servers**
- **AI စွမ်းအားသုံး ဆော့ဖ်ဝဲလ် ရေးသားခြင်း**
- **Extensions များ နှင့် အခြား IDE လုပ်ဆောင်ချက်များ**

စာသား တည်းဖြတ်ခြင်း နှင့် Vim

ပရိုဂရမ် ရေးသားသည့်အခါ သင်၏ အချိန်အများစုကို စာကြောင်းအရှည်ကြီးများ ဆက်တိုက် ရေးသားခြင်း သို့မဟုတ် ဖိုင်တစ်ခုလုံးကို အထက်မှ အောက်သို့ အဆုံးထိ ဖတ်ရှုခြင်းတို့ထက် ကုဒ်များအတွင်း သွားလာလှုပ်ရှားခြင်း၊ ကုဒ် အစိတ်အပိုင်းများကို ဖတ်ရှုခြင်းနှင့် ကုဒ်များကို ပြင်ဆင်တည်းဖြတ်ခြင်းတို့တွင် ပိုမို ကုန်လွန်စေသည်။ **Vim** သည် ဤကဲ့သို့သော အလုပ်များကို ထိရောက်စွာ လုပ်ဆောင်နိုင်ရန် အထူး ပြုပြင်ဖန်တီးထားသည့် text editor တစ်ခု ဖြစ်သည်။

Vim ၏ အခြေခံ သဘောတရား။ Vim ၏ အခြေခံတွင် လှုပသော အတွေးအခေါ် တစ်ခု ရှိသည် - ၎င်း၏ interface ကိုယ်တိုင်က စာသားများကို သွားလာကြည့်ရှုရန်နှင့် ပြင်ဆင်တည်းဖြတ်ရန် ဒီဇိုင်းထုတ်ထားသည့် ပရိုဂရမ်းမင်း ဘာသာစကား တစ်ခု ဖြစ်သည်။ ခလုတ်နှိပ်ချက်များ (မှတ်မိလွယ်သော အမည်များပါဝင်သည့်) သည် command များ ဖြစ်ကြပြီး ဤ command များကို ပေါင်းစပ် အသုံးပြုနိုင်ပါသည်။ Vim သည် မောက်စ် (mouse) အသုံးပြုခြင်းကို ရှောင်ကြဉ်သည်၊ အကြောင်းမှာ မောက်စ်ကို သုံးခြင်းသည် နှေးကွေးလွန်းသောကြောင့် ဖြစ်သည်။ Vim သည် အကွာအဝေး အဝေးကြီး လှုပ်ရှားရမှုများသည့်အတွက် arrow keys များကိုပင် ရှောင်ကြဉ်သည်။ ရလဒ်အနေဖြင့် - ဦးနှောက်နှင့် ကွန်ပျူတာ တိုက်ရိုက် ချိတ်ဆက်ထားသကဲ့သို့ ခံစားရပြီး သင်၏ တွေးခေါ်မှု မြန်နှုန်းနှင့် အမှီ လိုက်နိုင်သည့် editor တစ်ခု ဖြစ်လာစေသည်။

အခြား ဆော့ဖ်ဝဲလ်များတွင် Vim ကို ထောက်ပံ့ပေးထားမှု။ Vim ၏ အဓိက အနှစ်သာရ အယူအဆများမှ အကျိုးကျေးဇူး ရရှိရန်အတွက် **Vim** ဆော့ဖ်ဝဲလ် သီးသန့်ကိုပဲ အသုံးပြုရန် မလိုပါ။ စာသား တည်းဖြတ်မှု ပါဝင်သည့် ပရိုဂရမ် အများအပြားသည် “Vim mode” ကို သီးသန့် plugin အဖြစ် သို့မဟုတ် အသင့်ပါဝင်သော လုပ်ဆောင်ချက်အဖြစ် ထောက်ပံ့ပေးထားကြသည်။ ဥပမာအားဖြင့် VS Code တွင် **VSCodeVim** (https://marketplace.visualstudio.com/items?itemName=vscodevim.vim) plugin ရှိပြီး၊ Zsh တွင် Vim emulation အတွက် **အသင့်ပါဝင်သည့် ထောက်ပံ့မှု** (https://zsh.sourceforge.io/Guide/zshguide04.html) ပါရှိသည်၊ ထို့ပြင် Claude Code တွင်ပင် Vim editor mode အတွက် **အသင့်ပါဝင်သည့် ထောက်ပံ့မှု** (https://code.claude.com/docs/en/interactive-mode#vim-editor-mode) ပါဝင်ပါသည်။ သင် အသုံးပြုသည့် စာသား တည်းဖြတ်မှု ပါဝင်သော မည်သည့် tool မဆို Vim mode ကို နည်းလမ်း တစ်ခုခုဖြင့် ထောက်ပံ့ပေးထားနိုင်ခြေ အလွန် များပါသည်။

Modal editing

Vim သည် *modal editor* တစ်ခု ဖြစ်သည် - လုပ်ဆောင်ရမည့် လုပ်ငန်း အမျိုးအစားများအပေါ် မူတည်၍ ကွဲပြားသော လုပ်ဆောင်မှု mode များ ပါရှိသည်။

- **Normal**: ဖိုင်အတွင်း သွားလာလှုပ်ရှားရန်နှင့် ပြင်ဆင်တည်းဖြတ်မှုများ ပြုလုပ်ရန်

- **Insert**: စာသားများ ရိုက်ထည့်ရန်
- **Replace**: စာသားများကို အစားထိုးရန်
- **Visual** (plain၊ line သို့မဟုတ် block): စာသား အစိတ်အပိုင်းများကို ရွေးချယ် (select) ရန်
- **Command-line**: command တစ်ခုကို Run ရန်

ခလုတ်နှိပ်ချက်များ (Keystrokes) သည် သက်ဆိုင်ရာ လုပ်ဆောင်မှု mode အပေါ် မူတည်၍ အဓိပ္ပာယ် ကွဲပြားသွားသည်။ ဥပမာအားဖြင့် Insert mode တွင် `x` စာလုံးကို နှိပ်ပါက “x” စာလုံးကိုသာ ရိုက်ထည့်ပေး မည် ဖြစ်သော်လည်း Normal mode တွင် နှိပ်ပါက cursor ရောက်နေသော စာလုံးကို ဖျက်ပစ်မည် ဖြစ်ပြီး Visual mode တွင်မူ ရွေးချယ်ထားသော စာသား (selection) ကို ဖျက်ပစ်မည် ဖြစ်သည်။

Vim ၏ မူလ ပုံသေ သတ်မှတ်ချက်တွင် လက်ရှိ ရောက်ရှိနေသည့် mode ကို အောက်ခြေ ဘယ်ဘက်ထောင့် တွင် ဖော်ပြပေးထားသည်။ စတင်အသုံးပြုချိန် မူလ mode မှာ Normal mode ဖြစ်သည်။ ပုံမှန်အားဖြင့် သင်၏ အချိန်အများစုကို Normal mode နှင့် Insert mode တို့အကြား ကူးပြောင်း အသုံးပြုရမည် ဖြစ်သည်။

မည်သည့် mode မှမဆို Normal mode သို့ ပြန်လည် ကူးပြောင်းရန် `<ESC>` (escape key) ကို နှိပ်ခြင်းဖြင့် mode ကို ပြောင်းလဲနိုင်သည်။ Normal mode မှနေ၍ Insert mode သို့ ဝင်ရောက်ရန် `i` ၊ Replace mode သို့ ဝင်ရောက်ရန် `R` ၊ Visual mode သို့ ဝင်ရောက်ရန် `v` ၊ Visual Line mode သို့ ဝင်ရောက်ရန် `V` ၊ Visual Block mode သို့ ဝင်ရောက်ရန် `<C-v>` (Ctrl-V၊ အချို့နေရာများတွင် `^V` ဟုလည်း ရေးသားကြသည်) နှင့် Command-line mode သို့ ဝင်ရောက်ရန် `:` ကို အသုံးပြုနိုင်သည်။

Vim အသုံးပြုချိန်တွင် `<ESC>` key ကို အကြိမ်အများပြား နှိပ်ရမည် ဖြစ်သဖြင့် Caps Lock key ကို Escape အဖြစ် ရေတွက်ရန် remapping လုပ်ခြင်း (**macOS အတွက် လမ်းညွှန်** (https://vim.fandom.com/wiki/Map_caps_lock_to_escape_in_macOS)) သို့မဟုတ် ရိုးရှင်းသော key အစီအစဉ် တစ်ခုဖြင့် `<ESC>` အတွက် **အစားထိုး mapping ရေးဆွဲခြင်း** (https://vim.fandom.com/wiki/Avoid_the_escape_key#Mappings) တို့ကို ပြုလုပ်ထားရန် စဉ်းစားပါ။

အခြေခံများ- စာသား ရိုက်ထည့်ခြင်း

Normal mode မှနေ၍ Insert mode သို့ ဝင်ရောက်ရန် `i` ကို နှိပ်ပါ။ ၎င်းနောက်တွင် `<ESC>` ကို နှိပ်၍ Normal mode သို့ မပြန်မချင်း Vim သည် အခြားသော text editor များကဲ့သို့ လုပ်ဆောင်နေမည် ဖြစ်သည်။ ထိုလုပ်ဆောင်ချက်နှင့် အထက်တွင် ရှင်းပြခဲ့သော အခြေခံများသည် Vim ဖြင့် ဖိုင်များကို စတင် ပြင်ဆင် တည်းဖြတ်ရန် လိုအပ်သမျှ ဖြစ်ပါသည် (သို့သော် သင်၏ အချိန်အားလုံးကို Insert mode ထဲ၌သာ ကုန်လွန် နေပါက ပိုမို ထိရောက်မှု ရှိမည်တော့ မဟုတ်ပါ)။

Vim ၏ interface သည် ပရိုဂရမ်းမင်း ဘာသာစကား တစ်ခု ဖြစ်သည်

Vim ၏ interface သည် ပရိုဂရမ်းမင်း ဘာသာစကား တစ်ခု ဖြစ်သည်။ ခလုတ်နှိပ်ချက်များ (မှတ်မိလွယ်သော အမည်များပါရှိသော) သည် command များ ဖြစ်ကြပြီး ဤ command များကို *တွဲဖက် ပေါင်းစပ်* နိုင်ပါသည်။ ၎င်းသည် မြန်ဆန် ထိရောက်သော ရွှေ့လျားမှုများနှင့် ပြင်ဆင်မှုများကို ပြုလုပ်နိုင်စေသည်၊ အထူးသဖြင့် keyboard ၏ ခလုတ် တည်နေရာများကို မှတ်မိသွားချိန်တွင် စာရိုက်ခြင်းက အလွန် မြန်ဆန် လာသကဲ့သို့ ထို command များသည် muscle memory ဖြစ်လာသည့်အခါ အလွန် ထိရောက် လာမည် ဖြစ်သည်။

ရွှေ့လျားခြင်း (Movement)

သင်၏ အချိန်အများစုကို Normal mode တွင် ကုန်လွန်စေပြီး ဖိုင်အတွင်း သွားလာ ရွှေ့လျားရန်အတွက် movement command များကို အသုံးပြုသင့်သည်။ Vim တွင် ရွှေ့လျားမှုများကို “နာမ်များ” (nouns) ဟု လည်း ခေါ်ဆိုကြသည်၊ အကြောင်းမှာ ၎င်းတို့သည် စာသား အစိတ်အပိုင်းများကို ညွှန်းဆိုသောကြောင့် ဖြစ်သည်။

- အခြေခံ ရွှေ့လျားမှု: `hjkl` (ဘယ်၊ အောက်၊ အထက်၊ ညာ)
- စကားလုံးများ: `w` (နောက်စကားလုံး)၊ `b` (စကားလုံး၏ အစ)၊ `e` (စကားလုံး၏ အဆုံး)
- စာကြောင်းများ: `0` (စာကြောင်း၏ အစ)၊ `^` (ပထမဆုံး စာလုံးပါသော နေရာ)၊ `$` (စာကြောင်း၏ အဆုံး)
- မျက်နှာပြင်: `H` (မျက်နှာပြင် အထက်ပိုင်း)၊ `M` (မျက်နှာပြင် အလယ်ပိုင်း)၊ `L` (မျက်နှာပြင် အောက်ပိုင်း)
- စခရင် ရွှေ့ခြင်း (Scroll): `Ctrl-u` (အထက်သို့)၊ `Ctrl-d` (အောက်သို့)
- ဖိုင်: `gg` (ဖိုင်၏ အစ)၊ `G` (ဖိုင်၏ အဆုံး)
- စာကြောင်း နံပါတ်များ: `:{number}<CR>` သို့မဟုတ် `{number}G` (စာကြောင်း နံပါတ် {number})

```
- `<CR>` သည် carriage return / enter key ကို ညွှန်းဆိုသည်
```

- အထွေထွေ: `%` (ကွင်းစ/ကွင်းပိတ် သို့မဟုတ် တွန့်ကွင်း ကဲ့သို့သော ကိုက်ညီသည့် ပစ္စည်း)
- ရှာဖွေမှု (Find): `f{character}`၊ `t{character}`၊ `F{character}`၊ `T{character}`

```
- လက်ရှိ စာကြောင်းပေါ်တွင် {character} ကို ရှေ့သို့/နောက်သို့ ရှာရန် သို့မဟုတ် ထိုနေရာအထိ ရွှေ့ရန်
- ကိုက်ညီသော နေရာများအကြား သွားလာရန် `,` / `;` ကို သုံးသည်
```

- ရှာဖွေမှု (Search): `/{regex}` ၊ ကိုက်ညီသော နေရာများအကြား သွားလာရန် `n` / `N` ကို သုံးသည်

ရွေးချယ်ခြင်း (Selection)

Visual mode များ-

- Visual: `v`
- Visual Line: `V`
- Visual Block: `Ctrl-v`

စာသားများကို ရွေးချယ် (selection) ပြုလုပ်ရန် movement key များကို အသုံးပြုနိုင်သည်။

ပြင်ဆင်တည်းဖြတ်ခြင်း (Edits)

ယခင်က မောက်စ်ဖြင့် ပြုလုပ်ခဲ့သမျှ အရာအားလုံးကို ယခုအခါ movement command များ နှင့် ပေါင်းစပ်ထားသည့် editing command များ သုံး၍ keyboard ဖြင့် ပြုလုပ်နိုင်သည်။ ဤနေရာတွင် Vim ၏ interface သည် ပရိုဂရမ်းမင်း ဘာသာစကား တစ်ခုနှင့် စတင် တူညီလာသည်။ Vim ၏ editing command များကို “tool အပြုအမူများ” (verbs) ဟုလည်း ခေါ်ဆိုကြသည်၊ အကြောင်းမှာ tool အပြုအမူများသည် နာမ်များ (nouns) အပေါ် သက်ရောက် လုပ်ဆောင်သောကြောင့် ဖြစ်သည်။

- `i` Insert mode သို့ ဝင်ရောက်ရန်

```
- သို့သော် စာသားများကို ပြုပြင်ရန်/ဖျက်ရန်အတွက် backspace ထက် ပိုမို ထိရောက်သော အရာများကို အသုံးပြုလိုမည် ဖြစ်သည်
```

- `o` / `O` အောက်တွင် / အထက်တွင် စာကြောင်းအသစ် ရိုက်ထည့်ရန်
- `d{motion}` {motion} ကို ဖျက်ရန်

```
- ဥပမာ `dw` သည် စကားလုံးကို ဖျက်ခြင်း ဖြစ်သည်၊ `d$` သည် စာကြောင်း အဆုံးထိ ဖျက်ခြင်း ဖြစ်သည်၊ `d0` သည် စာကြောင်း အစထိ ဖျက်ခြင်း ဖြစ်သည်
```

- `c{motion}` {motion} ကို ပြောင်းလဲရန်

```
- ဥပမာ `cw` သည် စကားလုံးကို ပြောင်းလဲခြင်း ဖြစ်သည်
- `d{motion}` ပြီးနောက် `i` ကို နှိပ်သကဲ့သို့ ဖြစ်သည်
```

- `x` စာလုံး တစ်လုံး ဖျက်ရန် (`dl` နှင့် တူညီသည်)
- `s` စာလုံး တစ်လုံး အစားထိုးရန် (`cl` နှင့် တူညီသည်)
- Visual mode + ပြုပြင်ဖန်တီးမှု

```
- စာသားကို ရွေးချယ်ပြီး ဖျက်ရန် `d` ကို သို့မဟုတ် ပြောင်းလဲရန် `c` ကို နှိပ်ပါ
```

- Undo ပြုလုပ်ရန် `u` ၊ Redo ပြုလုပ်ရန် `<C-r>`
- ကူးယူရန် / “yank” လုပ်ရန် `y` (`d` ကဲ့သို့သော အခြား command အချို့သည်လည်း ကူးယူပေးသည်)
- ကူးထည့် (paste) ရန် `p`
- လေ့လာရန် အခြား အရာများစွာ ရှိသေးသည်: ဥပမာအားဖြင့် `~` သည် စာလုံး၏ စာလုံးကြီး/စာလုံးသေး (case) ကို ပြောင်းလဲပေးပြီး `J` သည် စာကြောင်းများကို ပေါင်းစပ်ပေးသည်

အရေအတွက်များ (Counts)

နာမ်များ (nouns) နှင့် tool အပြုအမူများ (verbs) ကို အရေအတွက် (count) တစ်ခုနှင့် ပေါင်းစပ် သုံးနိုင်ပြီး၊ ၎င်းသည် သတ်မှတ်ထားသော လုပ်ဆောင်ချက်ကို အရေအတွက် အကြိမ်အရေအတွက်အတိုင်း လုပ်ဆောင်ပေးမည် ဖြစ်သည်။

- `3w` ရှေ့သို့ စကားလုံး ၃ လုံး ရွှေ့ရန်
- `5j` အောက်သို့ စာကြောင်း ၅ ကြောင်း ရွှေ့ရန်
- `7dw` စကားလုံး ၇ လုံး ဖျက်ရန်

ပြုပြင်မွမ်းမံမှုများ (Modifiers)

နာမ်တစ်ခု၏ အဓိပ္ပာယ်ကို ပြောင်းလဲရန် modifiers များကို အသုံးပြုနိုင်သည်။ modifiers အချို့မှာ “အတွင်း ပိုင်း” သို့မဟုတ် “အထဲ၌” ဟု အဓိပ္ပာယ်ရသော `i` နှင့် “အပြင်ဘက် အပါအဝင်” သို့မဟုတ် “ပတ်လည်” ဟု အဓိပ္ပာယ်ရသော `a` တို့ ဖြစ်ကြသည်။

- `ci(` လက်ရှိ ကွင်းစ/ကွင်းပိတ် အတွင်းရှိ ပါဝင်အကြောင်းအရာများကို ပြောင်းလဲရန်
- `ci<a href="https://en.wikipedia.org/wiki/Fizz_buzz" class="ext-link">` လက်ရှိ လေးထောင့်ကွင်း အတွင်းရှိ ပါဝင်အကြောင်းအရာများကို ပြောင်းလဲရန်
- `da'` Single-quoted string တစ်ခုကို ဘေးပတ်လည် single-quote များ အပါအဝင် ဖျက်ပစ်ရန်

အားလုံးကို ပေါင်းစပ် အသုံးပြုခြင်း

ဤနေရာတွင် မှားယွင်းနေသော [fizz buzz (https://en.wikipedia.org/wiki/Fizz_buzz) ရေးသားချက် တစ်ခု ရှိသည်-

```
def fizz_buzz(limit):
    for i in range(limit):
        if i % 3 == 0:
            print("fizz", end="")
        if i % 5 == 0:
            print("fizz", end="")
        if i % 3 and i % 5:
            print(i, end="")
        print()

def main():
    fizz_buzz(20)
```

Normal mode မှ စတင်၍ ပြဿနာများကို ပြင်ဆင်ရန် အောက်ပါ command အစီအစဉ်အတိုင်း အသုံးပြုကြ ပါမည်-

- Main ကို မည်သည့်အခါမျှ ခေါ်ဆိုထားခြင်း မရှိပါ

```
- ဖိုင်၏ အဆုံးသို့ သွားရန် `G` ကို နှိပ်ပါ
- အောက်တွင် စာကြောင်းအသစ် **ဖွင့်** ရန် `o` ကို နှိပ်ပါ
- `if __name__ == "__main__": main()` ဟု ရိုက်ထည့်ပါ
    - သင့် editor တွင် Python ဘာသာစကား ထောက်ပံ့မှု ပါရှိပါက Insert mode တွင် အလိုအလျော
က် indentation ပြုလုပ်ပေးပေလိမ့်မည်
- Normal mode သို့ ပြန်သွားရန် `<ESC>` ကို နှိပ်ပါ
```

- 1 အစား 0 မှ စတင်နေပါသည်

```
- "range" ကို ရှာဖွေရန် `/` ပြီးနောက် `range` နှင့် `<CR>` ကို ရိုက်ထည့်ပါ
- ရှေ့သို့ စကားလုံး **၂ လုံး** ရွှေ့ရန် `ww` ကို နှိပ်ပါ (`2w` ကိုလည်း အသုံးပြုနိုင်သော်လည်း လက်တွ
ေ့တွင် အရေအတွက် အနည်းငယ်အတွက် count လုပ်ဆောင်ချက်ကို သုံးမည့်အစား key ကို ထပ်ခါထပ်ခါ
နှိပ်လေ့ ရှိကြသည်)
```

- insert mode သို့ ပြောင်းရန် `i` ကို နှိပ်ပြီး `1,` ကို ပေါင်းထည့်ပါ

```
- Normal mode သို့ ပြန်သွားရန် `<ESC>` ကို နှိပ်ပါ
- နောက်စကားလုံး၏ **အဆုံး** သို့ သွားရန် `e` ကို နှိပ်ပါ
- စာသား စတင် **နောက်ဆက်တွဲ ပေါင်းထည့်** ရန် `a` ကို နှိပ်ပြီး `+ 1` ကို ပေါင်းထည့်ပါ
- Normal mode သို့ ပြန်သွားရန် `<ESC>` ကို နှိပ်ပါ
```

- 5 ၏ ဆောက်တွဲများအတွက် “fizz” ကို ရိုက်နှိပ်နေပါသည်

```
- စာကြောင်း ၆ သို့ သွားရန် `:6<CR>` ကို နှိပ်ပါ
- '**"***' ၏ **အတွင်းပိုင်းကို ပြောင်းလဲ** ရန် `ci"` ကို နှိပ်ပြီး `"buzz"` သို့ ပြောင်းလဲပါ
- Normal mode သို့ ပြန်သွားရန် `<ESC>` ကို နှိပ်ပါ
```

Vim ကို လေ့လာခြင်း

Vim ကို လေ့လာရန် အကောင်းဆုံး နည်းလမ်းမှာ အခြေခံများ (ယခုအချိန်အထိ ကျွန်တော်တို့ ဆွေးနွေးခဲ့ပြီးသော အကြောင်းအရာများ) ကို လေ့လာပြီးနောက် သင်အသုံးပြုသည့် ဆော့ဖ်ဝဲလ် အားလုံးတွင် Vim mode ကို ဖွင့်ပြီး လက်တွေ့ စတင် အသုံးပြုခြင်း ဖြစ်သည်။ မောက်စ် သို့မဟုတ် arrow key များကို အသုံးပြုလိုသည့် သွေးဆောင်မှုကို ရှောင်ကြဉ်ပါ၊ အချို့သော editor များတွင် အလေ့အကျင့်ကောင်းများ စွဲမြဲစေရန် arrow key များကို unbind ပြုလုပ်ထားနိုင်သည်။

ထပ်ဆောင်း အရင်းအမြစ်များ

- ယခင် တန်းခွဲမှ [Vim သင်ခန်းစာ](https://missing-semester-my.github.io/2020/editors/) — ထိုနေရာတွင် Vim အကြောင်းကို ပိုမို အသေးစိတ် ဆွေးနွေးထားပါသည်
- `vimtutor` သည် Vim နှင့်အတူ တပါတည်း ပါဝင်လာသည့် လမ်းညွှန် သင်ခန်းစာ ဖြစ်သည် — Vim တပ်ဆင်ထားပါက သင့် shell မှနေ၍ `vimtutor` ကို Run နိုင်ရပါမည်
- [Vim Adventures](https://vim-adventures.com/) သည် Vim ကို လေ့လာနိုင်သော ဂိမ်းတစ်ခု ဖြစ်သည်
- [Vim Tips Wiki](https://vim.fandom.com/wiki/Vim_Tips_Wiki)
- [Vim Advent Calendar](https://vimways.org/2019/) တွင် Vim ဆိုင်ရာ အကြံပြုချက် အမျိုးမျိုး ပါဝင်သည်
- [VimGolf](https://www.vimgolf.com/) သည် [code golf](https://en.wikipedia.org/wiki/Code_golf) ပုံစံ ဖြစ်သော်လည်း ပရိုဂရမ်းမင်း ဘာသာစကား နေရာတွင် Vim ၏ UI ကို အသုံးပြုထားခြင်း ဖြစ်သည်
- [Vi/Vim Stack Exchange](https://vi.stackexchange.com/)
- [Vim Screencasts](http://vimcasts.org/)
- [Practical Vim](https://pragprog.com/titles/dnvim2/) (စာအုပ်)

Code intelligence နှင့် language servers

IDEs များသည် ပုံမှန်အားဖြင့် **Language Server Protocol** (https://microsoft.github.io/language-server-protocol/) ကို အကောင်အထည်ဖော်ထားသည့် *language servers* များနှင့် ချိတ်ဆက်ထားသော IDE extensions များမှ တစ်ဆင့် ကုဒ်၏ သဘောတရားဆိုင်ရာ နားလည်မှုကို လိုအပ်သည့် သက်ဆိုင်ရာ ပရိုဂရမ်းမင်း ဘာသာစကား အလိုက် ထောက်ပံ့မှုများကို ပေးအပ်ကြသည်။ ဥပမာအားဖြင့် **VS Code အတွက် Python extension** (https://marketplace.visualstudio.com/items?itemName=ms-python.python) သည် **Pylance** (https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance) ပေါ်တွင် အမှီပြုထားပြီး၊ **VS Code အတွက် Go extension** (https://marketplace.visualstudio.com/items?itemName=golang.go) သည် သီးသန့်ထုတ်လုပ်ထားသော **gopls** (https://go.dev/gopls/) ပေါ်တွင် အမှီပြုထားသည်။ သင် အသုံးပြုသည့် ဘာသာစကားများအတွက် extension နှင့် language server များကို တပ်ဆင်ခြင်းဖြင့် သင့် IDE တွင် ဘာသာစကားဆိုင်ရာ လုပ်ဆောင်ချက် အများအပြားကို အသုံးပြုနိုင်မည် ဖြစ်သည်၊ ဥပမာ-

- **Code completion။** `object.` ဟု ရိုက်ပြီးသည့်နောက် object ၏ fields နှင့် methods များကို မြင်တွေ့နိုင်ခြင်းကဲ့သို့သော ပိုမိုကောင်းမွန်သည့် autocomplete နှင့် autosuggest များ။
- **Inline documentation။** စာသားပေါ် mouse တင်ထားချိန် (hover) သို့မဟုတ် autosuggest ချိန်တွင် Documentation ကို မြင်တွေ့ရခြင်း။
- **Jump-to-definition။** အသုံးပြုထားသည့် နေရာမှ သတ်မှတ်ထားသည့် နေရာသို့ တိုက်ရိုက် သွားရောက်နိုင်ခြင်း၊ ဥပမာ field ညွှန်းဆိုချက် `object.field` မှ ၎င်း field ၏ definition သို့ သွားရောက်နိုင်ခြင်း။
- **Find references။** အထက်ပါ အချက်၏ ပြောင်းပြန် ဖြစ်ပြီး၊ field သို့မဟုတ် type ကဲ့သို့သော သီးခြား item တစ်ခုကို ညွှန်းဆိုထားသည့် နေရာ အားလုံးကို ရှာဖွေပေးခြင်း။
- **Help with imports။** Imports များကို စနစ်တကျ စီစဉ်ခြင်း၊ မလိုအပ်သော imports များကို ဖယ်ရှားခြင်း၊ လိုအပ်နေသော imports များကို အလံပြ အသိပေးခြင်း။
- **Code quality။** ဤ tools များကို သီးသန့် အသုံးပြုနိုင်သော်လည်း ဤလုပ်ဆောင်ချက်ကို language servers များကလည်း ပေးအပ်လေ့ ရှိသည်။ Code formatting သည် ကုဒ်များကို အလိုအလျောက် Indent နှင့် Format လုပ်ပေးပြီး၊ type checkers နှင့် linters များသည် စာရိုက်နေစဉ် ကုဒ်အတွင်းရှိ အမှားများကို ရှာဖွေပေးသည်။ ကျွန်တော်တို့သည် ဤလုပ်ဆောင်ချက် အကွာအဝေးကို **code quality ဆိုင်ရာ သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/code-quality/) တွင် ပိုမို အသေးစိတ် ဆွေးနွေးသွားပါမည်။

Language servers များကို ပြင်ဆင် သတ်မှတ်ခြင်း

အချို့သော ဘာသာစကားများအတွက် extension နှင့် language server တို့ကို တပ်ဆင်လိုက်ရုံဖြင့် အားလုံး အဆင်သင့် ဖြစ်သွားမည် ဖြစ်သည်။ အခြားသော ဘာသာစကားများအတွက်မူ language server ထံမှ အပြည့်အဝ အကျိုးကျေးဇူး ရရှိရန် သင့် IDE ထံသို့ သင့် ပတ်ဝန်းကျင် (environment) အကြောင်း ပြောပြ ပေးရန် လိုအပ်သည်။ ဥပမာအားဖြင့် VS Code ကို သင့် **Python environment** (https://code.visualstudio.com/docs/python/environments) ထို့သို့ ညွှန်ပြပေးခြင်းဖြင့် language server ကို သင် တပ်ဆင်ထားသော packages များကို မြင်တွေ့နိုင်စေမည် ဖြစ်သည်။ Environments အကြောင်းကို ကျွန်တော်တို့၏ **code များကို ထုပ်ပိုးခြင်း နှင့် ဖြန့်ဝေခြင်း ဆိုင်ရာ သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/shipping-code/) တွင် ပိုမို အသေးစိတ် ဆွေးနွေးထားပါသည်။

ဘာသာစကားအပေါ်မူတည်၍ သင့် language server အတွက် ပြင်ဆင် သတ်မှတ်နိုင်သည့် settings အချို့ ရှိနိုင်သည်။ ဥပမာအားဖြင့် VS Code တွင် Python ထောက်ပံ့မှုကို အသုံးပြု၍ Python ၏ optional type annotations များကို အသုံးမပြုသော project များအတွက် static type checking ကို ပိတ်ထားနိုင်သည်။

AI စွမ်းအားသုံး ဆော့ဖ်ဝဲလ် ရေးသားခြင်း

၂၀၂၁ အလယ်ပိုင်းတွင် OpenAI ၏ **Codex model** (https://openai.com/index/openai-codex/) ကို အသုံးပြုထားသည့် **GitHub Copilot** စတင် မိတ်ဆက်ချိန်မှစ၍ **LLMs** (https://en.wikipedia.org/wiki/Large_language_model) များကို ဆော့ဖ်ဝဲလ် အင်ဂျင်နီယာ လောကတွင် ကျယ်ကျယ်ပြန့်ပြန့် အသုံးပြုလာကြသည်။ လက်ရှိတွင် အဓိက အသုံးပြုနေသည့် ပုံစံ ၃ မျိုး ရှိသည် - autocomplete၊ inline chat နှင့် coding agents တို့ ဖြစ်ကြသည်။

Autocomplete

AI စွမ်းအားသုံး autocomplete သည် သင့် IDE ရှိ ရိုးရာ autocomplete နှင့် ပုံစံတူပင် ဖြစ်ပြီး သင် စာရိုက်နေစဉ် cursor ရောက်ရှိနေသည့် နေရာတွင် ဖြည့်စွက်ချက်များကို အကြံပြုပေးသည်။ အချို့ အချိန်များတွင် ၎င်းကို "အလိုလို အဆင်ပြေနေသည့်" သာမန် လုပ်ဆောင်ချက် တစ်ခုအဖြစ် အသုံးပြုကြသည်။ ထို့ပြင် AI autocomplete ကို ပုံမှန်အားဖြင့် ကုဒ်မှတ်ချက်များ (code comments) အသုံးပြု၍ **prompt ပေးကြသည်** (https://en.wikipedia.org/wiki/Prompt_engineering)။

ဥပမာအားဖြင့် ဤသင်ခန်းစာ မှတ်စုများ၏ ပါဝင်အကြောင်းအရာများကို Download ရယူပြီး လင့်ခ်များ အားလုံးကို ထုတ်ယူပေးမည့် script တစ်ခု ရေးသားကြပါစို့။ ကျွန်တော်တို့အနေဖြင့် အောက်ပါအတိုင်း စတင် ရေးသားနိုင်သည်-

```
import requests

def download_contents(url: str) -> str:
```

Model သည် function ၏ အဓိက အစိတ်အပိုင်းကို အလိုအလျောက် ဖြည့်စွက်ပေးမည် ဖြစ်သည်-

```
    response = requests.get(url)
    return response.text
```

ကျွန်တော်တို့သည် မှတ်ချက်များကို အသုံးပြု၍ ဖြည့်စွက်ချက်များကို ထပ်မံ လမ်းညွှန်ပေးနိုင်သည်။ ဥပမာ အားဖြင့် Markdown လင့်ခ်များ အားလုံးကို ထုတ်ယူရန် function တစ်ခု စတင်ရေးသားသော်လည်း ၎င်းတွင် လုံလောက်သော အမည်ဖော်ပြချက် မရှိသေးပါက-

```
def extract(contents: str) -> list[str]:
```

Model သည် အောက်ပါအတိုင်း ဖြည့်စွက်ပေးပေလိမ့်မည်-

```
    lines = contents.splitlines()
    return [line for line in lines if line.strip()]
```

ကျွန်တော်တို့သည် ကုဒ် မှတ်ချက်များမှတစ်ဆင့် ဖြည့်စွက်မှုကို လမ်းညွှန်ပေးနိုင်သည်-

```
def extract(content: str) -> list[str]:
    # extract all Markdown links from the content
```

ဤတစ်ကြိမ်တွင် Model သည် ပိုမိုကောင်းမွန်သော ဖြည့်စွက်မှုကို ပေးအပ်သည်-

```
    import re
    pattern = r'\[.*?\]\((.*?)\)'
    return re.findall(pattern, content)
```

ဤနေရာတွင် ဤ AI coding tool ၏ အားနည်းချက် တစ်ခုကို မြင်တွေ့နိုင်သည်- ၎င်းသည် cursor နေရာတွင်သာ ဖြည့်စွက်ချက်များကို ပေးနိုင်ခြင်း ဖြစ်သည်။ ဤဖြစ်ရပ်တွင် `import re` ကို function အတွင်း၌ ထည့်သွင်းမည့်အစား module အဆင့်တွင် ထားရှိခြင်းက ပိုမိုကောင်းမွန်သော အလေ့အကျင့် ဖြစ်ပေလိမ့်မည်။

အထက်ပါ ဥပမာသည် ကုဒ်မှတ်ချက်များ သုံး၍ code completion ကို မည်သို့ လမ်းညွှန်နိုင်သည်ကို ပြသရန် အတွက် နာမည် သေချာ မပေးထားသော function ကို အသုံးပြုထားခြင်း ဖြစ်သည်၊ လက်တွေ့တွင် သင် အနေဖြင့် `extract_links` ကဲ့သို့သော ပိုမို ရှင်းလင်းစွာ အမည်ပေးထားသည့် function များဖြင့် ရေးသားလိုမည် ဖြစ်ပြီး docstring များကိုလည်း ရေးသားလိုမည် ဖြစ်သည် (ထိုအပေါ်မူတည်၍ model သည် အထက်ပါအတိုင်း တူညီသော ဖြည့်စွက်ချက်ကို ထုတ်ပေးရမည် ဖြစ်သည်)။

စမ်းသပ်ပြသရန် အလို့ငှာ ကျွန်တော်တို့ script ကို အပြီးသတ် ရေးသားနိုင်သည်-

```
print(extract(download_contents("https://raw.githubusercontent.com/missing-seme
ster/missing-semester/refs/heads/master/_2026/development-environment.md")))
```

Inline chat

Inline chat သည် စာကြောင်း သို့မဟုတ် အစိတ်အပိုင်းတစ်ခုကို ရွေးချယ်ပြီး ပြင်ဆင်မှု ပြုလုပ်ရန် AI model ထံသို့ တိုက်ရိုက် prompt ပေးနိုင်စေသည်။ ဤ တို့ပြန်မှု mode တွင် model သည် ရှိပြီးသား ကုဒ်များကို ပြင်ဆင်ပေးနိုင်သည် (cursor ၏ နောက်ပိုင်းမှ ကုဒ်များကိုသာ ဖြည့်စွက်ပေးသည့် autocomplete နှင့် ကွဲပြားသည်)။

အထက်ပါ ဥပမာကို ဆက်လက် လေ့လာရလျှင် ပြင်ပမှ `requests` library ကို မသုံးရန် ဆုံးဖြတ်လိုက်သည် ဆိုပါစို့။ သက်ဆိုင်ရာ စာကြောင်း ၃ ကြောင်းကို ရွေးချယ်၍ inline chat ကို ခေါ်ယူပြီး အောက်ပါအတိုင်း ပြောဆိုနိုင်သည်-

```
use built-in libraries instead
```

Model မှ အောက်ပါအတိုင်း အဆိုပြု ပေးပါမည်-

```
from urllib.request import urlopen

def download_contents(url: str) -> str:
    with urlopen(url) as response:
        return response.read().decode('utf-8')
```

Coding agents

Coding agents အကြောင်းကို **Agentic Coding သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/agentic-coding/) တွင် ပိုမို အသေးစိတ် ဆွေးနွေးထားပါသည်။

အကြံပြုထားသော ဆော့ဖ်ဝဲလ်များ

လူကြိုက်များသော AI IDEs အချို့မှာ **GitHub Copilot** extension ပါဝင်သည့် **VS Code** နှင့် **Cursor** (https://cursor.com/) တို့ ဖြစ်ကြသည်။ GitHub Copilot ကို လက်ရှိတွင် ကျောင်းသားများ၊ ဆရာ/မများနှင့် လူကြိုက်များသော open source project များမှ ပြုပြင်ထိန်းသိမ်းသူများအတွက် **အခမဲ့ ရရှိနိုင်ပါသည်** (https://github.com/education/students)။ ဤနယ်ပယ်သည် အလွန် လျင်မြန်စွာ တိုးတက်ပြောင်းလဲနေသော နယ်ပယ် ဖြစ်သည်။ ထိပ်တန်း ထုတ်ကုန် အများအပြားတွင် အကြမ်းအားဖြင့် တူညီသော လုပ်ဆောင်ချက်များ ပါရှိကြသည်။

Extensions များ နှင့် အခြား IDE လုပ်ဆောင်ချက်များ

IDEs များသည် စွမ်းဆောင်ရည်မြင့်မားသော tools များ ဖြစ်ကြပြီး *extensions* များ ကြောင့် ပိုမို စွမ်းဆောင်ရည် မြင့်မားလာကြသည်။ ကျွန်တော်တို့အနေဖြင့် သင်ခန်းစာ တစ်ခုတည်းတွင် ဤလုပ်ဆောင်ချက် အားလုံးကို မဖော်ပြနိုင်သော်လည်း လူကြိုက်များသော extension အချို့အတွက် ညွှန်ပြချက် အချို့ကို ဤနေရာတွင် ပေးအပ်ထားပါသည်။ ဤနယ်ပယ်ကို ကိုယ်တိုင် စူးစမ်း လေ့လာရန် တိုက်တွန်းပါသည်၊ အွန်လိုင်းတွင် Vim plugins များအတွက် **Vim Awesome** (https://vimawesome.com/) နှင့် **လူကြိုက်အများဆုံး စီစဉ်ထားသော VS Code extensions များ** (https://marketplace.visualstudio.com/search?target=VSCode&category=All%20categories&sortBy=Installs) ကဲ့သို့သော လူကြိုက်များသည့် IDE extensions စာရင်းများစွာ ရှိပါသည်။

- **Development containers** (https://containers.dev/): လူကြိုက်များသော IDEs များမှ ထောက်ပံ့ပေးထားပြီး (ဥပမာ **VS Code မှ ထောက်ပံ့ပေးထားမှု** (https://code.visualstudio.com/docs/devcontainers/containers))၊ dev containers သည် သင့်အား development tools များကို Run ရန် container တစ်ခုကို အသုံးပြုနိုင်စေသည်။ ၎င်းသည် ရွှေ့ပြောင်းရလွယ်ကူမှု (portability) သို့မဟုတ် သီးခြားခွဲထုတ်ထားမှု (isolation) အတွက် အသုံးဝင်နိုင်သည်။ **code များကို ထုပ်ပိုးခြင်း နှင့် ဖြန့်ဝေခြင်း ဆိုင်ရာ သင်ခန်းစာ** (https://missing-semester-my.github.io/2026/shipping-code/) တွင် containers အကြောင်းကို ပိုမို အသေးစိတ် ဖော်ပြထားပါသည်။
- Remote development: SSH ကို အသုံးပြု၍ အဝေးထိန်း စက် (remote machine) ပေါ်တွင် ဆော့ဖ်ဝဲလ် ရေးသားခြင်း (ဥပမာ **VS Code အတွက် Remote SSH plugin** (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh) ဖြင့်)။ ဥပမာအားဖြင့် Cloud ပေါ်ရှိ စွမ်းဆောင်ရည်မြင့် GPU စက်ပေါ်တွင် ကုဒ်များ ရေးသားပြီး Run လိုသည့်အခါ ဤအရာသည် အသုံးဝင်နိုင်သည်။
- Collaborative editing: Google Docs ပုံစံအတိုင်း ဖိုင်တစ်ခုတည်းကို အတူတကွ တည်းဖြတ်ခြင်း (ဥပမာ **VS Code အတွက် Live Share plugin** (https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare) ဖြင့်)။

လေ့ကျင့်ခန်းများ

1. သင့် editor နှင့် သင့် shell ကဲ့သို့သော Vim mode ထောက်ပံ့သည့် သင်အသုံးပြုနေသော ဆော့ဖ်ဝဲလ် အားလုံးတွင် Vim mode ကို ဖွင့်ပြီး ရှေ့လာမည့် လအတွင်း သင့် စာသား တည်းဖြတ်မှု အားလုံးအတွက် Vim mode ကို အသုံးပြုပါ။ တစ်စုံတစ်ခု လုပ်ဆောင်ရာတွင် မထိရောက်ဟု ခံစားရပါက သို့မဟုတ် “ပိုမို ကောင်းမွန်သော နည်းလမ်း ရှိရမည်” ဟု တွေးမိပါက Google တွင် ရှာဖွေကြည့်ပါ၊ ပိုမိုကောင်းမွန်သော နည်းလမ်း ရှိနေနိုင်ပါသည်။
2. **VimGolf** (https://www.vimgolf.com/) မှ စိန်ခေါ်မှု တစ်ခုကို အပြီးသတ်ပါ။
3. သင် လုပ်ဆောင်နေသော project တစ်ခုအတွက် IDE extension နှင့် language server ကို ပြင်ဆင် သတ်မှတ်ပါ။ Library dependencies များအတွက် jump-to-definition ကဲ့သို့သော မျှော်လင့်ထားသည့် လုပ်ဆောင်ချက် အားလုံး မှန်ကန်စွာ အလုပ်လုပ်ကြောင်း သေချာပါစေ။ ဤလေ့ကျင့်ခန်းအတွက် သုံးနိုင်သော ကုဒ်မရှိပါက GitHub မှ open-source project တစ်ခုခုကို အသုံးပြုနိုင်ပါသည် (ဤ **project** (https://github.com/spf13/cobra) ကဲ့သို့သော)။
4. IDE extensions စာရင်းကို လေ့လာပြီး သင့်အတွက် အသုံးဝင်မည့် extension တစ်ခုကို တပ်ဆင်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. tmux	https://github.com/tmux/tmux	
2. Vim	https://www.vim.org/	
3. Zsh	https://www.zsh.org/	
4. Ruff	https://docs.astral.sh/ruff/	
5. Mypy	https://mypy-lang.org/	
6. VSCodeVim	https://marketplace.visualstudio.com/items?itemName=vscodevim.vim	
7. အသင့်ပါဝင်သည့် ထောက်ပံ့မှု	https://zsh.sourceforge.io/Guide/zshguide04.html	
8. အသင့်ပါဝင်သည့် ထောက်ပံ့မှု	https://code.claude.com/docs/en/interactive-mode#vim-editor-mode	
9. macOS အတွက် လမ်းညွှန်	https://vim.fandom.com/wiki/Map_caps_lock_to_escape_in_macOS	
10. အစားထိုး mapping ရေးဆွဲခြင်း	https://vim.fandom.com/wiki/Avoid_the_escape_key#Mappings	
11. `လက်ှီ လေးထောင့်ကွင်းအတွင်းရှိ ပါဝင်အကြောင်းအရာများကို ပြောင်းလဲရန် - `da'` Single-quoted string တစ်ခုကို ဘေးပတ်လည် single-quote များအပါအဝင် မျက်ပစ်ရန် ## အားလုံးကို ပေါင်းစပ် အသုံးပြုခြင်း ဤနေရာတွင် မှားယွင်းနေသော [fizz buzz	https://en.wikipedia.org/wiki/Fizz_buzz	
12. Vim သင်ခန်းစာ	https://missing-semester-my.github.io/2020/editors/	
13. Vim Adventures	https://vim-adventures.com/	
14. Vim Tips Wiki	https://vim.fandom.com/wiki/Vim_Tips_Wiki	

Resource / Description	URL	Micro QR
15. Vim Advent Calendar	https://vimways.org/2019/	
16. VimGolf	https://www.vimgolf.com/	
17. code golf	https://en.wikipedia.org/wiki/Code_golf	
18. Vi/Vim Stack Exchange	https://vi.stackexchange.com/	
19. Vim Screencasts	http://vimcasts.org/	
20. Practical Vim	https://pragprog.com/titles/dnvim2/	
21. Language Server Protocol	https://microsoft.github.io/language-server-protocol/	
22. VS Code အတွက် Python extension	https://marketplace.visualstudio.com/items?itemName=ms-python.python	
23. Pylance	https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance	
24. VS Code အတွက် Go extension	https://marketplace.visualstudio.com/items?itemName=golang.go	
25. gopls	https://go.dev/gopls/	
26. code quality ဆိုင်ရာ သင်ခန်းစာ	https://missing-semester-my.github.io/2026/code-quality/	
27. Python environment	https://code.visualstudio.com/docs/python/environments	
28. code များကို ထုပ်ပိုးခြင်း နှင့် ဖြန့်ဝေခြင်း ဆိုင်ရာ သင်ခန်းစာ	https://missing-semester-my.github.io/2026/shipping-code/	
29. Codex model	https://openai.com/index/openai-codex/	
30. LLMs	https://en.wikipedia.org/wiki/Large_language_model	

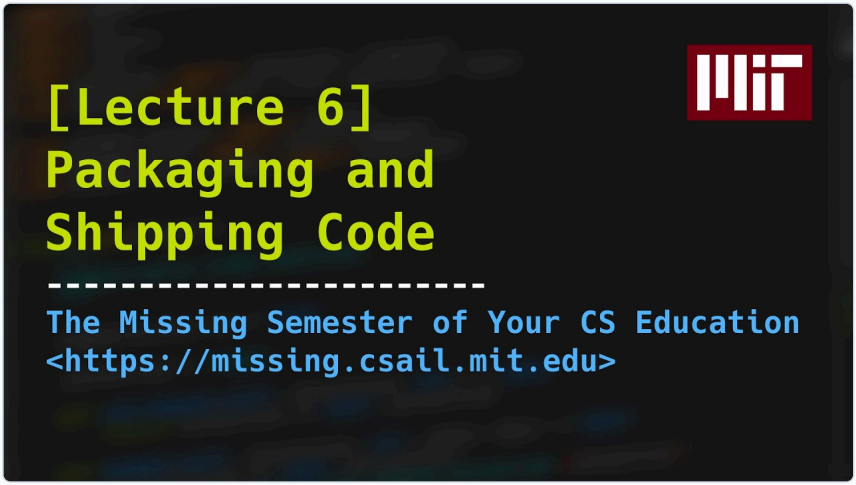
Resource / Description	URL	Micro QR
31. prompt ပေးကြသည်	https://en.wikipedia.org/wiki/Prompt_engineering	
32. Agentic Coding သင်ခန်းစာ	https://missing-semester-my.github.io/2026/agentic-coding/	
33. Cursor	https://cursor.com/	
34. အခမဲ့ ရရှိနိုင်ပါသည်	https://github.com/education/students	
35. Vim Awesome	https://vimawesome.com/	
36. လူကြိုက်အများဆုံး စီစဉ်ထားသော VS Code extensions များ	https://marketplace.visualstudio.com/search?target=VSCode&category=All%20categories&sortBy=Installs	
37. Development containers	https://containers.dev/	
38. VS Code မှ ထောက်ပံ့ပေးထားမှု	https://code.visualstudio.com/docs/devcontainers/containers	
39. VS Code အတွက် Remote SSH plugin	https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh	
40. VS Code အတွက် Live Share plugin	https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare	
41. ဤ project	https://github.com/spf13/cobra	

Packaging and Shipping Code

အနှစ်ချုပ် - ပရောဂျက် ထုပ်ပိုးခြင်း (packaging)၊ ပတ်ဝန်းကျင်များ (environments)၊ ဗားရှင်း သတ်မှတ်ခြင်း (versioning)၊ နှင့် libraries၊ applications၊ ဝန်ဆောင်မှုများကို တပ်ဆင်ဖြန့်ဖြူးခြင်း (deploying) တို့အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Packaging and Shipping Code



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=KBMiB-8P4Ns

ကုဒ်တစ်ခုကို ရည်ရွယ်ချက်အတိုင်း အလုပ်လုပ်အောင် ရေးသားရန် ခက်ခဲပါသည်၊ ထိုကုဒ်ကိုပင် မိမိစက်နှင့် မတူသော အခြားစက်တစ်ခုပေါ်တွင် အလုပ်လုပ်အောင် ပြုလုပ်ရန်မှာ ပို၍ပင် ခက်ခဲလေ့ရှိပါသည်။

ကုဒ်များကို Shipping ပြုလုပ်ခြင်း (ဖြန့်ဖြူးထုတ်လုပ်ခြင်း) ဆိုသည်မှာ သင်ရေးသားထားသော ကုဒ်ကို မိမိကွန်ပျူတာ၏ တိကျသော Setup မလိုဘဲ အခြားသူတစ်ဦးဦးက အလွယ်တကူ Run နိုင်သော အသုံးပြုနိုင်သည့် ပုံစံသို့ ပြောင်းလဲပေးခြင်း ဖြစ်ပါသည်။ ကုဒ်များကို Shipping ပြုလုပ်ရာတွင် ပုံစံအမျိုးမျိုး ရှိနိုင်ပြီး Programming language၊ System library များ၊ Operating system စသည့် အချက်အလက်များစွာပေါ်တွင် မူတည်ပါသည်။ ၎င်းသည် သင်တည်ဆောက်နေသည့် အရာပေါ်တွင်လည်း မူတည်ပါသည်—Software library တစ်ခု၊ Command line tool တစ်ခု၊ သို့မဟုတ် Web service တစ်ခု စသည်တို့တွင် လိုအပ်ချက်များနှင့် Deployment အဆင့်များ အသီးသီး ကွဲပြားကြပါသည်။ မည်သို့ပင်ဖြစ်စေ၊ ဤအခြေအနေအားလုံးတွင် ဘုံတူညီသော ပုံစံတစ်ခု ရှိပါသည်—၎င်းမှာ ပေးအပ်ရမည့် အရာ (deliverable) သို့မဟုတ် *artifact* ဆိုသည်မှာ ဘာလဲဆိုသည်ကို သတ်မှတ်ရန်နှင့် ၎င်းသည် ၎င်း၏ ပတ်ဝန်းကျင် (environment) ပေါ်တွင် မည်သည့် ယူဆချက်များ ထားရှိသည်ဆိုသည်ကို သတ်မှတ်ပေးရန် လိုအပ်ခြင်း ဖြစ်ပါသည်။

ဤသင်ခန်းစာတွင် အောက်ပါတို့အကြောင်း လွှမ်းခြုံဆွေးနွေးသွားပါမည်-

- **မှီခိုမှုများနှင့် ပတ်ဝန်းကျင်များ (Dependencies & Environments)**
- **Artifact များနှင့် ထုပ်ပိုးခြင်း (Artifacts & Packaging)**
- **ထုတ်လုပ်မှုများနှင့် ဗားရှင်းသတ်မှတ်ခြင်း (Releases & Versioning)**
- **ထပ်မံဖန်တီးနိုင်စွမ်း (Reproducibility)**
- **VM များနှင့် Containers များ (VMs & Containers)**
- **စနစ်ပြင်ဆင်မှု (Configuration)**
- **ဝန်ဆောင်မှုများနှင့် စီမံခန့်ခွဲမှု (Services & Orchestration)**
- **ထုတ်ဝေခြင်း (Publishing)**

ပိုမိုနားလည်လွယ်ကူစေရန်အတွက် လက်တွေ့ကျသော ဥပမာများဖြစ်သည့် Python ecosystem ထဲမှ ဥပမာများဖြင့် ဤသဘောတရားများကို ရှင်းပြသွားပါမည်။ အခြား programming language ecosystem များအတွက် tool များ ကွဲပြားနိုင်သော်လည်း သဘောတရားအများစုမှာ အတူတူပင် ဖြစ်ပါသည်။

Dependencies & Environments

ခေတ်သစ် ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်မှုတွင် အဆင့်ဆင့် သီးခြားခွဲထုတ်ထားသော အလွှာများ (layers of abstraction) သည် နေရာတိုင်းတွင် ရှိနေပါသည်။ ပရိုဂရမ်များသည် အခြားlibraries သို့မဟုတ် ဝန်ဆောင်မှုများထံသို့ လုပ်ဆောင်ချက်များကို သဘာဝအတိုင်း လွှဲပြောင်းလုပ်ဆောင်လေ့ ရှိကြပါသည်။ သို့သော် ၎င်းသည် သင်၏ ပရိုဂရမ်နှင့် အလုပ်လုပ်ရန် လိုအပ်သော librariesအကြား *မှီခိုမှု (dependency)* ဆက်ဆံရေးတစ်ခုကို ပေါ်ပေါက်စေပါသည်။ ဥပမာအားဖြင့် Python တွင် ဝဘ်ဆိုက်တစ်ခု၏ အကြောင်းအရာကို ဆွဲယူရန် အတွက် ကျွန်ုပ်တို့သည် အောက်ပါအတိုင်း ပြုလုပ်လေ့ရှိကြသည်-

```
import requests

response = requests.get("https://missing.csail.mit.edu")
```

သို့သော် `requests` librarysသည် Python runtime နှင့်အတူ တစ်ပါတည်း ပါဝင်မလာပါ၊ ထို့ကြောင့် `requests` ကို Install မလုပ်ဘဲ ဤကုဒ်ကို Run ရန် ကြိုးပမ်းပါက Python သည် Error ပြသပါလိမ့်မည်-

```
$ python fetch.py
Traceback (most recent call last):
  File "fetch.py", line 1, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
```

ဤlibraryကို အသုံးပြုနိုင်ရန်အတွက် ကျွန်ုပ်တို့သည် ၎င်းကို install ပြုလုပ်ရန် ပထမဦးစွာ `pip install requests` ကို Run ရန် လိုအပ်ပါသည်။ `pip` ဆိုသည်မှာ Package များကို install ပြုလုပ်ရန်အတွက် Python programming language က ပံ့ပိုးပေးထားသော command line tool ဖြစ်ပါသည်။ `pip install requests` ကို စတင်ပတ်မောင်းလိုက်ပါက အောက်ပါ အဆင့်အတန်းအတိုင်း ဆက်တိုက် လုပ်ဆောင်သွားပါသည်-

1. Python Package Index (**PyPI** (https://pypi.org/)) တွင် requests ကို ရှာဖွေခြင်း
2. ကျွန်ုပ်တို့ အသုံးပြုနေသော platform အတွက် သင့်လျော်သော artifact ကို ရှာဖွေခြင်း
3. မှီခိုမှုများကို ဖြေရှင်းခြင်း (Resolve dependencies) — `requests` libraryကိုယ်တိုင်ကလည်း အခြား package များကို မှီခိုနေသဖြင့် installer သည် ဆက်စပ်မှီခိုနေသော (transitive dependencies) package များ၏ ကိုက်ညီသော ဗားရှင်းအားလုံးကို ရှာဖွေပြီး မတိုင်မီ ကြိုတင် install လုပ်ပေးရပါမည်

4. Artifact များကို Download ပြုလုပ်ပြီးနောက် ဖိုင်များကို ကျွန်ုပ်တို့၏ filesystem အတွင်းရှိ မှန်ကန်သော နေရာများသို့ ဖြေညှို့၍ ကူးယူထည့်သွင်းခြင်း

```
$ pip install requests
Collecting requests
  Downloading requests-2.32.3-py3-none-any.whl (64 kB)
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.4.0-cp311-cp311-manylinux_x86_64.whl (142 k
B)
Collecting idna<4,>=2.5
  Downloading idna-3.10-py3-none-any.whl (70 kB)
Collecting urllib3<3,>=1.21.1
  Downloading urllib3-2.2.3-py3-none-any.whl (126 kB)
Collecting certifi>=2017.4.17
  Downloading certifi-2024.8.30-py3-none-any.whl (167 kB)
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requ
ests
Successfully installed certifi-2024.8.30 charset-normalizer-3.4.0 idna-3.10 req
uests-2.32.3 urllib3-2.2.3
```

ဤနေရာတွင် `requests` ၌ `certifi` သို့မဟုတ် `charset-normalizer` ကဲ့သို့သော မိမိကိုယ်ပိုင် dependency များ ရှိနေပြီး `requests` ကို install မပြုလုပ်မီ ၎င်းတို့ကို အရင် install လုပ်ရမည်ကို တွေ့မြင်နိုင်ပါသည်။ Install လုပ်ပြီးသွားပါက Python runtime သည် import ပြုလုပ်သည့်အခါ ဤlibraryကို ရှာဖွေတွေ့ရှိနိုင်ပြီ ဖြစ်ပါသည်။

```
$ python -c 'import requests; print(requests.__path__)'
['/usr/local/lib/python3.11/dist-packages/requests']

$ pip list | grep requests
requests        2.32.3
```

Programming languages များတွင် librariesကို install ပြုလုပ်ခြင်းနှင့် ထုတ်ဝေခြင်း (publishing) တို့အတွက် မတူညီသော tool များ၊ စံနှုန်းများနှင့် အလေ့အထများ ရှိကြပါသည်။ Rust ကဲ့သို့သော အချို့ language များတွင် toolchain ကို တစ်ခုတည်းအဖြစ် ပေါင်းစည်းထားပါသည်— `cargo` က ထုတ်လုပ်ခြင်း (building)၊ စမ်းသပ်ခြင်း (testing)၊ dependency စီမံခန့်ခွဲခြင်း နှင့် ထုတ်ဝေခြင်းတို့အားလုံးကို ကိုင်တွယ်ပေးပါသည်။ Python ကဲ့သို့သော အခြား language များတွင်မူ ပေါင်းစည်းမှုသည် စံသတ်မှတ်ချက်အဆင့် (specification level) တွင် ဖြစ်ပေါ်ပါသည်—tool တစ်ခုတည်း မဟုတ်ဘဲ packaging မည်သို့ အလုပ်လုပ်သည်ကို သတ်မှတ်ပေးသည့် စံသတ်မှတ်ချက်များ ရှိနေသဖြင့် အလုပ်တစ်ခုစီအတွက် ယှဉ်ပြိုင်နေသော tool အများအပြားကို အသုံးပြုနိုင်စေပါသည် (`pip` သို့မဟုတ် **`uv`** (https://docs.astral.sh/uv/)၊ `setuptools`

သို့မဟုတ် `hatch` (https://hatch.pypa.io/) သို့မဟုတ် `poetry` (https://python-poetry.org/) စသည်ဖြင့်)။ LaTeX ကဲ့သို့သော အချို့ ecosystem များတွင်မူ TeX Live သို့မဟုတ် MacTeX ကဲ့သို့သော distribution များတွင် package ထောင်ပေါင်းများစွာကို ကြိုတင် install ပြုလုပ်၍ တစ်ပါတည်း ထည့်သွင်းပေးထားပါသည်။

Dependency များကို စတင်အသုံးပြုလာသည်နှင့်အမျှ Dependency ပဋိပက္ခများ (dependency conflicts) လည်း ပေါ်ပေါက်လာပါသည်။ ပရိုဂရမ်များသည် တူညီသော dependency ၏ ကိုက်ညီမှုမရှိသော ဗားရှင်းများကို တောင်းဆိုသည့်အခါ ပဋိပက္ခများ ဖြစ်ပေါ်တတ်ပါသည်။ ဥပမာအားဖြင့် `tensorflow==2.3.0` က `numpy>=1.16.0,<1.19.0` ကို လိုအပ်ပြီး `pandas==1.2.0` က `numpy>=1.16.5` ကို လိုအပ်ပါက `numpy>=1.16.5,<1.19.0` ကို ပြည့်မီသော မည်သည့်ဗားရှင်းမဆို ကိုက်ညီမည် ဖြစ်ပါသည်။ သို့သော် သင့် ပရောဂျက်ရှိ အခြား package တစ်ခုက `numpy>=1.19` ကို လိုအပ်နေပါက ကန့်သတ်ချက်အားလုံးကို ပြည့်မီသည့် ကိုက်ညီသော ဗားရှင်းမရှိတော့သဖြင့် ပဋိပက္ခ ဖြစ်ပေါ်လာမည် ဖြစ်ပါသည်။

package အများအပြားက မျှဝေသုံးစွဲထားသော dependency များ၏ အပြန်အလှန် ကိုက်ညီမှုမရှိသော ဗားရှင်းများကို တောင်းဆိုနေသည့် ဤအခြေအနေကို လေ့ရှိသောအားဖြင့် *dependency hell* ဟု ခေါ်ဆိုကြပါသည်။ ပဋိပက္ခများကို ကိုင်တွယ်ဖြေရှင်းနည်း တစ်ခုမှာ ပရိုဂရမ်တစ်ခုစီ၏ dependency များကို ၎င်းတို့၏ သီးခြား *environment* အတွင်းသို့ ခွဲထုတ်ထားခြင်း (isolate) ဖြစ်ပါသည်။ Python တွင် ကျွန်ုပ်တို့သည် အောက်ပါအတိုင်း Run ခြင်းဖြင့် virtual environment တစ်ခုကို ဖန်တီးပါသည်-

```
$ which python
/usr/bin/python
$ pwd
/home/missingsemester
$ python -m venv venv
$ source venv/bin/activate
$ which python
/home/missingsemester/venv/bin/python
$ which pip
/home/missingsemester/venv/bin/pip
$ python -c 'import requests; print(requests.__path__)'
['/home/missingsemester/venv/lib/python3.11/site-packages/requests']

$ pip list
Package Version
------- -------
pip     24.0
```

Environment တစ်ခုကို မိမိကိုယ်ပိုင် install လုပ်ထားသော package များ ပါဝင်သည့် သီးခြား ရပ်တည်နေသော language runtime တစ်ခုလုံးအဖြစ် တွေးတောနိုင်ပါသည်။ ဤ virtual environment သို့မဟုတ် venv သည် install လုပ်ထားသော dependency များကို မူလ global Python installation မှ ခွဲထုတ်ပေးပါသည်။ ပ

ရောဂျက်တစ်ခုစီအတွက် လိုအပ်သော dependency များကို သီးခြားထည့်သွင်းထားသည့် virtual environment တစ်ခု စီထားရှိခြင်းသည် ကောင်းမွန်သော အလေ့အထတစ်ခု ဖြစ်ပါသည်။

> ခေတ်သစ် operating system အများအပြားတွင် Python ကဲ့သို့သော programming language runtime များကို တစ်ပါတည်း ထည့်သွင်းပေးထားသော်လည်း OS ကိုယ်တိုင်က ယင်းတို့၏ လုပ်ဆောင်ချက်များအတွက် ၎င်းတို့ကို အားကိုးနေရနိုင်သဖြင့် ထို မူလ installation များကို ပြင်ဆင်ခြင်း မပြုသင့်ပါ။ ထို့အစား သီးခြား environment များကို အသုံးပြုခြင်းကို ပိုမို ဦးစားပေးပါ။

အချို့သော language များတွင် installation protocol ကို tool တစ်ခုဖြင့် သတ်မှတ်ထားခြင်း မဟုတ်ဘဲ စံသတ်မှတ်ချက် (specification) အဖြစ် သတ်မှတ်ထားပါသည်။ Python တွင် **PEP 517** (https://peps.python.org/pep-0517/) က build system interface ကို သတ်မှတ်ပေးပြီး **PEP 621** (https://peps.python.org/pep-0621/) က ပရောဂျက်၏ metadata များကို `pyproject.toml` တွင် မည်သို့ သိမ်းဆည်းရမည်ကို ဖော်ပြထားပါသည်။ ၎င်းသည် developer များအား `pip` ထက် ပိုမို ကောင်းမွန်လာစေရန်နှင့် `uv` ကဲ့သို့ ပိုမို ပေါ့ပါးမြန်ဆန်သော tool များကို ဖန်တီးနိုင်စေရန် အထောက်အကူပြုခဲ့ပါသည်။ `uv` ကို install လုပ်ရန်အတွက် `pip install uv` ဟု လုပ်ဆောင်ရုံမျှဖြင့် လုံလောက်ပါသည်။

`pip` အစား `uv` ကို အသုံးပြုခြင်းသည် interface တူညီသော်လည်း သိသိသာသာ ပိုမို မြန်ဆန်ပါသည်-

```
$ uv pip install requests
Resolved 5 packages in 12ms
Prepared 5 packages in 0.45ms
Installed 5 packages in 8ms
 + certifi==2024.8.30
 + charset-normalizer==3.4.0
 + idna==3.10
 + requests==2.32.3
 + urllib3==2.2.3
```

> Install လုပ်သည့် အချိန်ကို အလွန်အမင်း လျှော့ချပေးနိုင်သောကြောင့် ဖြစ်နိုင်ပါက `pip` အစား `uv pip` ကို အသုံးပြုရန် အလွန် အကြံပြုအပ်ပါသည်။

Dependency များကို ခွဲထုတ်ပေးသည့်အပြင် Environment များသည် သင်၏ programming language runtime ၏ မတူညီသော ဗားရှင်းများကို သုံးနိုင်စေရန်လည်း ဆောင်ရွက်ပေးနိုင်ပါသည်။

```
$ uv venv --python 3.12 venv312
Using CPython 3.12.7
Creating virtual environment at: venv312

$ source venv312/bin/activate && python --version
Python 3.12.7

$ uv venv --python 3.11 venv311
Using CPython 3.11.10
Creating virtual environment at: venv311

$ source venv311/bin/activate && python --version
Python 3.11.10
```

မိမိ၏ ကုဒ်ကို Python ဗားရှင်း အများအပြားတွင် စမ်းသပ်ရန် လိုအပ်သည့်အခါ သို့မဟုတ် ပရောဂျက်တစ်ခုက တိကျသော ဗားရှင်းတစ်ခု လိုအပ်သည့်အခါ ဤအရာက အကူအညီဖြစ်စေပါသည်။

> အချို့သော programming language များတွင် ပရောဂျက်တစ်ခုစီသည် မိမိကိုယ်တိုင် ကိုယ်တိုင်ကိုယ်ကျ ဖန်တီးစရာ မလိုဘဲ ၎င်း၏ dependency များအတွက် environment တစ်ခုကို အလိုအလျောက် ရရှိကြသော်လည်း မူဝါဒမှာ အတူတူပင် ဖြစ်ပါသည်။ ယနေ့ခေတ် language အများစုတွင် စနစ်တစ်ခုတည်း၌ language ဗားရှင်း အများအပြားကို စီမံခန့်ခွဲနိုင်သည့် စနစ် ပါရှိကြပြီး ပရောဂျက်တစ်ခုစီအတွက် မည်သည့် ဗားရှင်းကို အသုံးပြုရမည်ဆိုသည်ကို သတ်မှတ်ပေးနိုင်ပါသည်။

Artifacts & Packaging

ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်မှုတွင် ကျွန်ုပ်တို့သည် မူရင်းကုဒ် (source code) နှင့် artifact များကို ခြားနားစွာ သတ်မှတ်ကြပါသည်။ Developer များသည် source code များကို ရေးသားကြပြီး ဖတ်ရှုကြသည်၊ artifact များမှာမူ ထို source code မှ ထုတ်လုပ်ထားသော၊ ထုပ်ပိုးထားသည့်၊ ဖြန့်ဖြူးနိုင်သော ရလဒ်များဖြစ်ပြီး install လုပ်ရန် သို့မဟုတ် deploy လုပ်ရန် အသင့်ဖြစ်နေသော အရာများ ဖြစ်ကြပါသည်။ Artifact တစ်ခုသည် ကျွန်ုပ်တို့ Run သော ကုဒ်ဖိုင်တစ်ခုကဲ့သို့ ရိုးရှင်းနိုင်သကဲ့သို့ application တစ်ခု၏ လိုအပ်သော အစိတ်အပိုင်းအားလုံး ပါဝင်သည့် Virtual Machine တစ်ခုလုံးကဲ့သို့ ရှုပ်ထွေးနိုင်ပါသည်။ ကျွန်ုပ်တို့၏ လက်ရှိ directory တွင် Python ဖိုင် `greet.py` ရှိသည့် ဤဥပမာကို ကြည့်ပါ-

```
$ cat greet.py
def greet(name):
    print(f"Hello, {name}!")

$ python -c "from greet import greet; greet('World')"
Hello, World!

$ cd /tmp
$ python -c "from greet import greet; greet('World')"
ModuleNotFoundError: No module named 'greet'
```

အခြား directory တစ်ခုသို့ ပြောင်းရွှေ့လိုက်သည်နှင့် import လုပ်ခြင်း မအောင်မြင်တော့ပါ၊ အကြောင်းမှာ Python သည် သတ်မှတ်ထားသော နေရာများတွင်သာ module များကို ရှာဖွေသောကြောင့် ဖြစ်ပါသည် (လက်ရှိ directory၊ install လုပ်ထားသော package များ၊ နှင့် `PYTHONPATH` တွင် ပါဝင်သည့် လမ်းကြောင်းများ)။ Packaging သည် ကုဒ်ကို သိရှိပြီးသား နေရာတစ်ခုသို့ install ပြုလုပ်ပေးခြင်းဖြင့် ဤပြဿနာကို ဖြေရှင်းပေးပါသည်။

Python တွင် library တစ်ခုကို packaging ပြုလုပ်ရာ၌ `pip` သို့မဟုတ် `uv` ကဲ့သို့သော package installer များက သက်ဆိုင်ရာ ဖိုင်များကို install လုပ်ရာတွင် အသုံးပြုနိုင်သည့် artifact တစ်ခုကို ထုတ်လုပ်ပေးခြင်း ပါဝင်ပါသည်။ Python artifact များကို *wheels* ဟု ခေါ်ဆိုပြီး package တစ်ခုကို install ရန် လိုအပ်သော အချက်အလက်များ အားလုံး ပါဝင်ပါသည်- ကုဒ်ဖိုင်များ၊ package ဆိုင်ရာ metadata များ (အမည်၊ ဗားရှင်း၊ dependency များ)၊ နှင့် environment ထဲရှိ မည်သည့်နေရာတွင် ဖိုင်များ ထည့်သွင်းရမည်ဆိုသည့် ညွှန်ကြားချက်များ။ Artifact တစ်ခုကို Build ပြုလုပ်ရန်အတွက် ပရောဂျက်၏ အသေးစိတ်အချက်အလက်များ၊ လိုအပ်သော dependency များ၊ package ၏ ဗားရှင်း နှင့် အခြား အချက်အလက်များကို ဖော်ပြထား

သည့် project file တစ်ခု (manifest ဟုလည်း ခေါ်လေ့ရှိသည်) ရေးသားရန် လိုအပ်ပါသည်။ Python တွင် ဤရည်ရွယ်ချက်အတွက် `pyproject.toml` ကို အသုံးပြုကြပါသည်။

> `pyproject.toml` သည် ခေတ်သစ် နည်းလမ်းဖြစ်ပြီး အကြံပြုထားသော နည်းလမ်းဖြစ်ပါသည်။ `requirements.txt` သို့မဟုတ် `setup.py` ကဲ့သို့သော ယခင် packaging နည်းလမ်းများကို ထောက်ပံ့ပေးနေဆဲ ဖြစ်သော်လည်း ဖြစ်နိုင်လျှင် `pyproject.toml` ကို ဦးစားပေး အသုံးပြုသင့်ပါသည်။

အောက်တွင် Command-line tool တစ်ခုကိုလည်း ပံ့ပိုးပေးထားသည့် library တစ်ခုအတွက် အနည်းဆုံး လိုအပ်သော `pyproject.toml` ဖိုင် ဥပမာ ဖြစ်ပါသည်-

```
[project]
name = "greeting"
version = "0.1.0"
description = "A simple greeting library"
dependencies = ["typer>=0.9"]

[project.scripts]
greet = "greeting:cli"

[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
```

`typer` librarysသည် သာမန် boilerplate များစွာ ရေးစရာ မလိုဘဲ command-line interface များကို ဖန်တီးရန်အတွက် လူကြိုက်များသော Python package တစ်ခု ဖြစ်ပါသည်။

၎င်းနှင့် သက်ဆိုင်သည့် `greeting.py` ဖိုင်မှာ အောက်ပါအတိုင်း ဖြစ်ပါသည်-

```
import typer


def greet(name: str) -> None:
    print(f"Hello, {name}!")


def cli() -> None:
    typer.run(greet)


if __name__ == "__main__":
    cli()
```

ဤဖိုင်ဖြင့် ယခုအခါ ကျွန်ုပ်တို့သည် wheel ကို build ပြုလုပ်နိုင်ပြီ ဖြစ်ပါသည်-

```
$ uv build
Building source distribution...
Building wheel from source distribution...
Successfully built dist/greeting-0.1.0.tar.gz
Successfully built dist/greeting-0.1.0-py3-none-any.whl

$ ls dist/
greeting-0.1.0-py3-none-any.whl
greeting-0.1.0.tar.gz
```

`.whl` ဖိုင်သည် wheel (သတ်မှတ်ထားသော တည်ဆောက်ပုံပါရှိသည့် zip archive) ဖြစ်ပြီး၊ `.tar.gz` မှာမူ မူရင်း source ထံမှ build ပြုလုပ်ရန် လိုအပ်သည့် စနစ်များအတွက် source distribution ဖိုင် ဖြစ်ပါသည်။

မည်သည့်အရာများ ထုပ်ပိုးသွားသည်ကို ကြည့်ရှုရန် wheel ၏ အကြောင်းအရာများကို စစ်ဆေးနိုင်ပါသည်။

```
$ unzip -l dist/greeting-0.1.0-py3-none-any.whl
Archive:  dist/greeting-0.1.0-py3-none-any.whl
  Length      Date    Time    Name
---------  ---------- -----   ----
      150  2024-01-15 10:30   greeting.py
      312  2024-01-15 10:30   greeting-0.1.0.dist-info/METADATA
       92  2024-01-15 10:30   greeting-0.1.0.dist-info/WHEEL
        9  2024-01-15 10:30   greeting-0.1.0.dist-info/top_level.txt
      435  2024-01-15 10:30   greeting-0.1.0.dist-info/RECORD
---------                     -------
      998                     5 files
```

ယခုအခါ ဤ wheel ကို အခြားသူတစ်ဦးထံ ပေးလိုက်ပါက ၎င်းတို့သည် အောက်ပါအတိုင်း Run ၍ install ပြုလုပ်နိုင်ပါသည်-

```
$ uv pip install ./greeting-0.1.0-py3-none-any.whl
$ greet Alice
Hello, Alice!
```

၎င်းသည် အစောပိုင်းက ကျွန်ုပ်တို့ build ပြုလုပ်ခဲ့သော libraryနှင့် `greet` cli tool အပါအဝင် ၎င်းတို့၏ environment ထဲသို့ install ပြုလုပ်ပေးမည် ဖြစ်ပါသည်။

ဤနည်းလမ်းတွင် ကန့်သတ်ချက်များ ရှိပါသည်။ အထူးသဖြင့် ကျွန်ုပ်တို့၏ librarysသည် GPU အရှိန်မြှင့်တင်ရန်အတွက် CUDA ကဲ့သို့သော platform အလိုက် သီးခြားဖြစ်သော librariesကို မှီခိုနေပါက ကျွန်ုပ်တို့၏ artifact သည် ထိုသီးခြားlibraries install လုပ်ထားသည့် စနစ်များတွင်သာ အလုပ်လုပ်မည်ဖြစ်ပြီး၊ မတူညီသော platform များ (Linux, macOS, Windows) နှင့် architecture များ (x86, ARM) အတွက် သီးခြား wheel များကို build ပြုလုပ်ရန် လိုအပ်နိုင်ပါသည်။

ဆော့ဖ်ဝဲလ်များကို install ပြုလုပ်ရာတွင် source မှ install ပြုလုပ်ခြင်းနှင့် ကြိုတင် build လုပ်ထားသော binary (prebuilt binary) မှ install ပြုလုပ်ခြင်းတို့အကြား အရေးကြီးသော ခြားနားချက် ရှိပါသည်။ Source မှ install ပြုလုပ်ခြင်း ဆိုသည်မှာ မူရင်းကုဒ်ကို Download ပြုလုပ်ပြီး မိမိစက်ပေါ်တွင် compile ပြုလုပ်ခြင်းကို ဆိုလိုပါသည်—၎င်းသည် မိမိစက်တွင် compiler နှင့် build tool များ ရှိနေရန် လိုအပ်ပြီး၊ ပရောဂျက်ကြီးများအတွက် အချိန်အတော်အတန် ယူရနိုင်ပါသည်။

Prebuilt binary ကို install ပြုလုပ်ခြင်းဆိုသည်မှာ အခြားသူတစ်ဦးက compile လုပ်ပြီးသား artifact ကို Download ပြုလုပ်ခြင်း ဖြစ်ပါသည်—ပိုမို မြန်ဆန်ပြီး ရိုးရှင်းသော်လည်း binary သည် သင့်စက်၏ platform နှင့် architecture တို့နှင့် ကိုက်ညီရပါမည်။ ဥပမာအားဖြင့် **ripgrep ၏ releases စာမျက်နှာ**

(https://github.com/BurntSushi/ripgrep/releases) တွင် Linux (x86_64, ARM)၊ macOS (Intel, Apple Silicon)၊ နှင့် Windows တို့အတွက် prebuilt binary များကို ပြသထားပါသည်။

Releases & Versioning

ကုဒ်များကို အစဉ်အမြဲ ဆက်တိုက် တည်ဆောက်နေကြသော်လည်း ရလဒ်များ (releases) ကိုမူ သီးခြား အချိန်ကာလအလိုက် ထုတ်ဝေကြပါသည်။ ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်မှုတွင် ဖွံ့ဖြိုးတိုးတက်ရေး environment (development environment) နှင့် ထုတ်လုပ်ရေး environment (production environment) တို့ အကြား ရှင်းလင်းသော ခြားနားချက် ရှိပါသည်။ ကုဒ်ကို prod စက်များထံသို့ *shipped* မပြုလုပ်မီ dev environment တွင် အလုပ်လုပ်ကြောင်း သက်သေပြရန် လိုအပ်ပါသည်။ Release ပြုလုပ်သည့် လုပ်ငန်းစဉ်တွင် စမ်းသပ်ခြင်း၊ dependency စီမံခန့်ခွဲခြင်း၊ ဗားရှင်းသတ်မှတ်ခြင်း၊ configuration စနစ်ပြင်ဆင်ခြင်း၊ deployment ပြုလုပ်ခြင်း နှင့် ထုတ်ဝေခြင်း အပါအဝင် အဆင့်များစွာ ပါဝင်ပါသည်။

Software library များသည် တည်ငြိမ်၍ ရပ်တန့်မနေဘဲ အမှားပြင်ဆင်ချက်များနှင့် လုပ်ဆောင်ချက်အသစ်များ ရရှိလာသည်နှင့်အမျှ အချိန်နှင့်အမျှ တိုးတက်ပြောင်းလဲလာကြပါသည်။ အချိန်ကာလတစ်ခုရှိ library၏ အခြေအနေနှင့် ကိုက်ညီသော သီးခြား ဗားရှင်း အမှတ်အသားများဖြင့် ဤတိုးတက်ပြောင်းလဲမှုကို ကျွန်ုပ်တို့ စောင့်ကြည့်မှတ်တမ်းတင်ကြပါသည်။ library တစ်ခု၏ မူလလုပ်ဆောင်ချက် ပြောင်းလဲမှုများသည် အရေးမကြီးသော လုပ်ဆောင်ချက်များကို ပြင်ဆင်သည့် patch များ၊ လုပ်ဆောင်ချက်များကို တိုးချဲ့ပေးသည့် feature အသစ်များ မှသည် နောက်ပြန်ဆုတ်မရသော ပြောင်းလဲမှုများ (breaking backwards compatibility) အထိ ရှိနိုင်ပါသည်။ Changelog များသည် ဗားရှင်းတစ်ခုက မည်သည့် ပြောင်းလဲမှုများကို မိတ်ဆက်ပေးသည်ဆိုသည်ကို မှတ်တမ်းတင်ထားပါသည်—ဤသည်မှာ သတင်းလွှာပုံစံဖြင့် ထုတ်ဝေမှုအသစ်နှင့် သက်ဆိုင်သည့် ပြောင်းလဲမှုများကို ဆက်သွယ်အကြောင်းကြားရန် ဆော့ဖ်ဝဲလ် developer များ အသုံးပြုသည့် Documentation များ ဖြစ်ကြပါသည်။

သို့သော် dependency တိုင်းရှိ ပြောင်းလဲမှုများကို အမြဲမပြတ် စောင့်ကြည့်မှတ်တမ်းတင်နေရန်မှာ လက်တွေ့မကျပါ၊ ကျွန်ုပ်တို့၏ dependency များ၏ dependency များဖြစ်သည့် transitive dependency များကို ထည့်သွင်းစဉ်းစားသည့်အခါ ပို၍ပင် ခက်ခဲပါလိမ့်မည်။

> သင့်ပရောဂျက်၏ dependency သစ်ပင်တစ်ခုလုံးကို package အားလုံးနှင့် ၎င်းတို့၏ transitive dependency များကို သစ်ပင်ပုံစံဖြင့် ပြသပေးသည့် `uv tree` ဖြင့် မြင်တွေ့နိုင်ပါသည်။

ဤပြဿနာကို ရိုးရှင်းစေရန်အတွက် ဆော့ဖ်ဝဲလ် ဗားရှင်း သတ်မှတ်နည်းဆိုင်ရာ စံနှုန်းများ ရှိကြပြီး၊ အထင်ရှားဆုံး တစ်ခုမှာ **Semantic Versioning** (https://semver.org/) သို့မဟုတ် SemVer ဖြစ်ပါသည်။ Semantic

Versioning အောက်တွင် ဗားရှင်းတစ်ခုခု၌ အကိန်းပြည့် တန်ဖိုးတစ်ခုစီ ယူသော MAJOR.MINOR.PATCH ပုံစံ ပါရှိသည့် အမှတ်အသားတစ်ခု ရှိပါသည်။ အတိုချုပ်အားဖြင့် အဆင့်မြှင့်တင်ရာတွင်-

- PATCH (ဥပမာ- 1.2.3 → 1.2.4) တွင် အမှားပြင်ဆင်ချက်များ (bug fixes) သာ ပါဝင်သင့်ပြီး နောက်ကြောင်းပြန် သဟဇာတ ဖြစ်မှု (backwards compatible) အပြည့်အဝ ရှိရပါမည်
- MINOR (ဥပမာ- 1.2.3 → 1.3.0) သည် နောက်ကြောင်းပြန် သဟဇာတဖြစ်သော နည်းလမ်းဖြင့် လုပ်ဆောင်ချက်အသစ်များကို ထည့်သွင်းပေးပါသည်
- MAJOR (ဥပမာ- 1.2.3 → 2.0.0) သည် ကုဒ်ပြင်ဆင်မှုများ လိုအပ်နိုင်သည့် ဘက်ပေါင်းစုံ ပြောင်းလဲမှုများ (breaking changes) ကို ညွှန်ပြပါသည်

> ဤသည်မှာ အတိုချုပ် ရှင်းပြချက်ဖြစ်ပြီး ဥပမာအားဖြင့် 0.1.3 မှ 0.2.0 သို့ ပြောင်းလဲခြင်းက အဘယ်ကြောင့် breaking change များ ဖြစ်ပေါ်စေနိုင်သည် သို့မဟုတ် 1.0.0-rc.1 က မည်သည့် အဓိပ္ပာယ်ဆောင်သည်ဆိုသည်ကို နားလည်ရန်အတွက် SemVer Specification အပြည့်အစုံကို ဖတ်ရှုရန် တိုက်တွန်းပါသည်။ Python packaging သည် semantic versioning ကို မူလကတည်းက ထောက်ပံ့ပေးထားသဖြင့် ကျွန်ုပ်တို့၏ dependency ဗားရှင်းများကို သတ်မှတ်သည့်အခါ သတ်မှတ်ကန့်သတ်ချက် အမျိုးမျိုးကို အသုံးပြုနိုင်ပါသည်။

`pyproject.toml` ဖိုင်တွင် ကျွန်ုပ်တို့၏ dependency များ၏ ကိုက်ညီသော ဗားရှင်းအပိုင်းအခြားများကို ကန့်သတ်ရန် မတူညီသော နည်းလမ်းများ ရှိကြပါသည်-

```
[project]
dependencies = [
    "requests==2.32.3",  # Exact version - ဤသီးခြားဗားရှင်းတစ်ခုတည်းသာ
    "click>=8.0",        # Minimum version - 8.0 သို့မဟုတ် ပိုသစ်သောဗားရှင်း
    "numpy>=1.24,<2.0",  # Range - အနည်းဆုံး 1.24 သို့သော် 2.0 ထက်ငယ်ရမည်
    "pandas~=2.1.0",     # Compatible release - >=2.1.0 နှင့် <2.2.0
]
```

ဗားရှင်း သတ်မှတ်ချက်ကန့်သတ်မှုများ (Version specifiers) သည် မတူညီသော သီးခြားအဓိပ္ပာယ်များဖြင့် package manager အများအပြား (npm, cargo စသည်) တွင် တည်ရှိကြပါသည်။ `~=` operator သည် Python ၏ “compatible release” operator ဖြစ်ပါသည်— `~=2.1.0` ဆိုသည်မှာ “2.1.0 နှင့် ကိုက်ညီမှုရှိသော မည်သည့် ဗားရှင်းမဆို” ဟု အဓိပ္ပာယ်ရပြီး `>=2.1.0` နှင့် `<2.2.0` သို့ ပြောင်းလဲအဓိပ္ပာယ်ထွက်ပါသည်။ ၎င်းသည် npm နှင့် cargo တို့ရှိ SemVer ၏ သဟဇာတဖြစ်မှုဆိုင်ရာ သဘောတရားကို လိုက်နာသည့် caret (`^`) operator နှင့် ကြမ်းဖျင်းအားဖြင့် တူညီပါသည်။

ဆော့ဖ်ဝဲလ် အားလုံးသည် semantic versioning ကို အသုံးပြုကြသည် မဟုတ်ပါ။ အခြား အသုံးများသော နည်းလမ်းတစ်ခုမှာ Calendar Versioning (CalVer) ဖြစ်ပြီး ဗားရှင်းများကို semantic အဓိပ္ပာယ်ထက် ထုတ်ဝေသည့် ရက်စွဲများပေါ်တွင် အခြေခံထားပါသည်။ ဥပမာအားဖြင့် Ubuntu သည် `24.04` (၂၀၂၄ ခုနှစ် ဧပြီလ) နှင့် `24.10` (၂၀၂၄ ခုနှစ် အောက်တိုဘာလ) ကဲ့သို့သော ဗားရှင်းများကို အသုံးပြုပါသည်။ CalVer သည် ကိုက်ညီမှုဆိုင်ရာ အချက်အလက်များကို အသိပေးခြင်း မရှိသော်လည်း ထုတ်ဝေမှုတစ်ခု မည်မျှ ပုံရိပ်ဟောင်းနေပြီဆိုသည်ကို အလွယ်တကူ သိရှိစေပါသည်။ နောက်ဆုံးအနေဖြင့် semantic versioning သည်လည်း အမှားကင်းစင်သည်တော့ မဟုတ်ပါ၊ တစ်ခါတစ်ရံ ပြုပြင်ထိန်းသိမ်းသူများသည် မတော်တဆ minor သို့မဟုတ် patch release များတွင် breaking change များကို မိတ်ဆက်မိတတ်ကြပါသည်။

ထပ်မံဖန်တီးနိုင်စွမ်း (Reproducibility)

ခေတ်သစ် ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်မှုတွင် သင်ရေးသားသော ကုဒ်သည် သိသာထင်ရှားလှသော အလွှာများစွာ (layers of abstraction) ပေါ်တွင် တည်ရှိနေပါသည်။ ၎င်းတို့တွင် သင်၏ programming language runtime၊ ပြင်ပlibraries (third party libraries)၊ operating system သို့မဟုတ် hardware ကိုယ်တိုင် ပါဝင်ပါသည်။ ဤအလွှာများအနက် မည်သည့်အလွှာတွင်မဆို ကွဲပြားမှုရှိပါက သင့်ကုဒ်၏ မူလလုပ်ဆောင်ချက်ကို ပြောင်းလဲသွားစေနိုင်သည် သို့မဟုတ် ရည်ရွယ်ထားသည့်အတိုင်း အလုပ်မလုပ်အောင် တားဆီးနိုင်ပါသည်။ ထို့အပြင် အောက်ခံ hardware တွင် ကွဲပြားမှုများ ရှိနေခြင်းကပင် ဆော့ဖ်ဝဲလ်ကို Shipping ပြုလုပ်နိုင်သည့် သင့်စွမ်းရည်ကို သက်ရောက်မှု ရှိစေပါသည်။

library တစ်ခုကို Pinning ပြုလုပ်ခြင်းဆိုသည်မှာ အပိုင်းအခြားတစ်ခု သတ်မှတ်ခြင်း (ဥပမာ `requests>=2.0`) မဟုတ်ဘဲ တိကျသော ဗားရှင်းတစ်ခုကို သတ်မှတ်ပေးခြင်း ဖြစ်ပါသည် (ဥပမာ `requests==2.32.3`)။

Package manager ၏ အလုပ်တစ်ခုမှာ dependency များ နှင့် transitive dependency များက ထောက်ပံ့ပေးထားသော ကန့်သတ်ချက်များအားလုံးကို ထည့်သွင်းစဉ်းစားပြီး ထိုကန့်သတ်ချက်များအားလုံးကို ပြည့်မီစေမည့် ကိုက်ညီသော ဗားရှင်းစာရင်းတစ်ခုကို ထုတ်လုပ်ပေးရန် ဖြစ်ပါသည်။ ထို တိကျသော ဗားရှင်းစာရင်းကို ထပ်မံဖန်တီးနိုင်စွမ်း (reproducibility) ရည်ရွယ်ချက်များအတွက် ဖိုင်တစ်ခုအဖြစ် သိမ်းဆည်းထားနိုင်သည်—ဤဖိုင်များကို *lock files* ဟု ခေါ်ဆိုကြပါသည်။

```
$ uv lock
Resolved 12 packages in 45ms

$ cat uv.lock | head -20
version = 1
requires-python = ">=3.11"

[[package]]
name = "certifi"
version = "2024.8.30"
source = { registry = "https://pypi.org/simple" }
sdist = { url = "https://files.pythonhosted.org/...", hash = "sha256:..." }
wheels = [
    { url = "https://files.pythonhosted.org/...", hash = "sha256:..." },
]
...
```

Dependency ဗားရှင်း သတ်မှတ်ခြင်းနှင့် reproducibility တို့ကို ကိုင်တွယ်ရာတွင် အဓိကကျသော ခြားနားချက်တစ်ခုမှာ libraries နှင့် applications/ဝန်ဆောင်မှုများ (applications/services) အကြား ခြားနားချက် ဖြစ်ပါသည်။ library တစ်ခုကို မိမိကိုယ်ပိုင် dependency များ ရှိကောင်းရှိနိုင်သည့် အခြားကုဒ်များက import ပြုလုပ်၍ အသုံးပြုရန် ရည်ရွယ်ထားသဖြင့် အလွန်အမင်း တင်းကျပ်သော ဗားရှင်းကန့်သတ်ချက်များ သတ်မှတ်ခြင်းသည် သုံးစွဲသူ၏ အခြား dependency များနှင့် ပဋိပက္ခများ ဖြစ်ပေါ်စေနိုင်ပါသည်။ ဆန့်ကျင်ဘက်အားဖြင့် applications သို့မဟုတ် ဝန်ဆောင်မှုများသည် ဆော့ဖ်ဝဲလ်၏ နောက်ဆုံးသုံးစွဲသူများ ဖြစ်ကြပြီး မကြာခဏဆိုသလို ၎င်းတို့၏ လုပ်ဆောင်ချက်များကို programming interface ထက် user interface သို့မဟုတ် API မှတစ်ဆင့် ထုတ်ဖော် ပြသလေ့ရှိကြပါသည်။ librariesအတွက် ကျယ်ပြန့်သော package ecosystem နှင့် ကိုက်ညီမှု အများဆုံး ရရှိစေရန် ဗားရှင်း အပိုင်းအခြားများ (version ranges) ကို သတ်မှတ်ခြင်းသည် ကောင်းမွန်သော အလေ့အထ ဖြစ်ပါသည်။ applicationsအတွက်မူ တိကျသော ဗားရှင်းများကို Pinning ပြုလုပ်ခြင်းက reproducibility ကို သေချာစေပါသည်—applicationကို မောင်းနှင်နေသူ တိုင်းသည် တူညီသော တိကျသော dependency များကို အသုံးပြုကြမည် ဖြစ်ပါသည်။

အမြင့်ဆုံး reproducibility လိုအပ်သော ပရောဂျက်များအတွက် **Nix** (https://nixos.org/) နှင့် **Bazel** (https://bazel.build/) ကဲ့သို့သော tool များသည် *hermetic* build များကို ပံ့ပိုးပေးပါသည်—ထိုတွင် compiler များ၊ system library များ၊ နှင့် build environment ကိုယ်တိုင်အပါအဝင် input တိုင်းကို pinning ပြုလုပ်ထားပြီး content-addressed ပြုလုပ်ထားပါသည်။ ၎င်းသည် build ကို မည်သည့်အချိန် သို့မဟုတ် မည်သည့်နေရာတွင်မဆို မောင်းနှင်သည်ဖြစ်စေ bit-for-bit အတူတူပင်ဖြစ်သော ရလဒ်များကို အာမခံပေးပါသည်။

> သင့်ကွန်ပျူတာ setup ၏ မိတ္တူအသစ်များကို လွယ်ကူစွာ စတင်နိုင်ရန်နှင့် ၎င်းတို့၏ စနစ်တစ်ခုလုံး ပြင်ဆင်ချက်များကို version-controlled ပြုလုပ်ထားသော configuration ဖိုင်များမှတစ်ဆင့် စီမံခန့်ခွဲနိုင်ရန် သင်၏ ကွန်ပျူတာ installation တစ်ခုလုံးကို စီမံရန်အတွက် NixOS ကိုပင် အသုံးပြုနိုင်ပါသည်။

ဆော့ဖ်ဝဲလ် ဖွံ့ဖြိုးတိုးတက်ရေးတွင် မပြီးဆုံးနိုင်သော တင်းမာမှုတစ်ခုမှာ ဆော့ဖ်ဝဲလ် ဗားရှင်းအသစ်များသည် ရည်ရွယ်ချက်ရှိရှိဖြစ်စေ မရည်ရွယ်ဘဲဖြစ်စေ ပျက်စီးမှုများကို မိတ်ဆက်ပေးလေ့ရှိပြီး၊ အခြားတစ်ဖက်တွင်မူ ဆော့ဖ်ဝဲလ် ဗားရှင်းအဟောင်းများသည် အချိန်နှင့်အမျှ လုံခြုံရေးဆိုင်ရာ အားနည်းချက်များ (security vulnerabilities) ကြုံတွေ့ရလေ့ရှိခြင်း ဖြစ်ပါသည်။ ကျွန်ုပ်တို့သည် ဤအရာကို ဆော့ဖ်ဝဲလ် ဗားရှင်းအသစ်များနှင့် ယှဉ်၍ applicationကို စမ်းသပ်သည့် continuous integration pipeline များ အသုံးပြုခြင်းဖြင့် လည်းကောင်း (**Code Quality and CI** (https://missing-semester-my.github.io/2026/code-quality/) သင်ခန်းစာတွင် ပိုမိုကြည့်ရှုပါမည်)၊ **Dependabot** (https://github.com/dependabot) ကဲ့သို့သော ကျွန်ုပ်တို့၏ dependency များ၏ ဗားရှင်းအသစ်များ ထွက်ရှိလာသည့်အခါ သိရှိနိုင်သည့် အလိုအလျောက် စနစ်များကို ထားရှိခြင်းဖြင့် လည်းကောင်း ဖြေရှင်းနိုင်ပါသည်။

CI စမ်းသပ်မှုများ ထားရှိသော်လည်း ဆော့ဖ်ဝဲလ် ဗားရှင်းများကို အဆင့်မြှင့်တင်သည့်အခါ dev နှင့် prod environment များအကြား ရှောင်လွှဲမရနိုင်သော ကွဲလွဲမှုများကြောင့် ပြဿနာများ ဖြစ်ပေါ်နေဆဲ ဖြစ်ပါသည်။ ထိုသို့သော အခြေအနေများတွင် အကောင်းဆုံး လုပ်ဆောင်ချက်မှာ ဗားရှင်း အဆင့်မြှင့်တင်မှုကို ပြန်လည် ရုပ်သိမ်းပြီး ၎င်းအစား သိရှိပြီးသား ကောင်းမွန်သော ဗားရှင်းကို ပြန်လည် deploy ပြုလုပ်သည့် *rollback* အစီအစဉ်တစ်ခု ထားရှိခြင်း ဖြစ်ပါသည်။

VM များနှင့် Containers များ (VMs & Containers)

ပိုမို ရှုပ်ထွေးသော dependency များကို အားကိုးလာသည်နှင့်အမျှ သင့်ကုဒ်၏ dependency များသည် package manager ကိုင်တွယ်နိုင်သည့် နယ်နိမိတ်ထက် ကျော်လွန်သွားနိုင်ခြေ ရှိပါသည်။ အသုံးများသော အကြောင်းအရင်းတစ်ခုမှာ သီးခြား system library များ သို့မဟုတ် hardware driver များနှင့် ချိတ်ဆက်လုပ်ဆောင်ရခြင်း ဖြစ်ပါသည်။ ဥပမာအားဖြင့် သိပ္ပံနည်းကျ တွက်ချက်မှုနှင့် AI တို့တွင် ပရိုဂရမ်များသည် GPU hardware ကို အသုံးပြုနိုင်ရန်အတွက် သီးသန့် librariesနှင့် driver များကို မကြာခဏ လိုအပ်ကြပါသည်။ System အဆင့် dependency အများအပြား (GPU driver များ၊ တိကျသော compiler ဗားရှင်းများ၊ OpenSSL ကဲ့သို့သော မျှဝေသုံးစွဲသည့် libraries) သည် စနစ်တစ်ခုလုံးအလိုက် (system-wide) install လုပ်ရန် လိုအပ်နေဆဲ ဖြစ်ပါသည်။

ရှေးယခင်က ဤပိုမိုကျယ်ပြန့်သော dependency ပြဿနာကို Virtual Machines (VMs) များဖြင့် ဖြေရှင်းခဲ့ကြပါသည်။ VM များသည် ကွန်ပျူတာတစ်လုံးလုံးကို abstraction ပြုလုပ်ပြီး မိမိကိုယ်ပိုင် သီးသန့် operating system ပါရှိသော လုံးဝ ခွဲထုတ်ထားသည့် environment တစ်ခုကို ပံ့ပိုးပေးပါသည်။ ပိုမို ခေတ်မီသော နည်းလမ်းမှာ container များ ဖြစ်ကြပြီး၊ ၎င်းတို့သည် ကွန်ပျူတာတစ်လုံးလုံးကို virtualize ပြုလုပ်ခြင်းထက် application တစ်ခုကို ၎င်း၏ dependency များ၊ libraries၊ နှင့် filesystem တို့နှင့်အတူ ထုပ်ပိုးထားသော်လည်း host ၏ operating system kernel ကို မျှဝေသုံးစွဲကြပါသည်။ Container များသည် kernel ကို မျှဝေသုံးစွဲသောကြောင့် VM များထက် ပေါ့ပါးပြီး စတင်ရလွယ်ကူကာ မောင်းနှင်ရာတွင် ပိုမို အလုပ်တွင် စွမ်းဆောင်ရည် ကောင်းမွန်ပါသည်။

လူကြိုက်အများဆုံး container platform မှာ **Docker** (https://www.docker.com/) ဖြစ်ပါသည်။ Docker သည် container များကို build ပြုလုပ်ခြင်း၊ ဖြန့်ဖြူးခြင်း နှင့် run ခြင်းတို့အတွက် စံသတ်မှတ်ထားသော နည်းလမ်းတစ်ခုကို မိတ်ဆက်ခဲ့ပါသည်။ အတွင်းပိုင်းတွင် Docker သည် containerd ကို ၎င်း၏ container runtime အဖြစ် အသုံးပြုပါသည်—၎င်းသည် Kubernetes ကဲ့သို့သော အခြား tool များပါ အသုံးပြုကြသည့် စက်မှုလုပ်ငန်းဆိုင်ရာ စံနှုန်းတစ်ခု ဖြစ်ပါသည်။

Container တစ်ခုကို run ခြင်းသည် ရိုးရှင်းပါသည်။ ဥပမာအားဖြင့် container တစ်ခုအတွင်း Python interpreter တစ်ခုကို မောင်းနှင်ရန် `docker run` ကို အသုံးပြုကြပါသည် (`-it` flag များသည် container ကို terminal ဖြင့် အပြန်အလှန် တုံ့ပြန်နိုင်စေရန် ပြုလုပ်ပေးပါသည်။ သင်ထွက်လိုက်သည့်အခါ container ရပ်တန့်သွားပါမည်။)။

```
$ docker run -it python:3.12 python
Python 3.12.7 (main, Nov  5 2024, 02:53:25) [GCC 12.2.0] on linux
>>> print("Hello from inside a container!")
Hello from inside a container!
```

လက်တွေ့တွင် သင့်ပရိုဂရမ်သည် filesystem တစ်ခုလုံးပေါ်တွင် မူတည်နေနိုင်ပါသည်။ ဤအရာကို ကျော်လွှားရန်အတွက် ကျွန်ုပ်တို့သည် application၏ filesystem တစ်ခုလုံးကို artifact အဖြစ် Shipping ပြုလုပ်ပေးသည့် container image များကို အသုံးပြုနိုင်ပါသည်။ Container image များကို ပရိုဂရမ်နည်းလမ်းဖြင့် ဖန်တီးကြပါသည်။ Docker ဖြင့် ကျွန်ုပ်တို့သည် Dockerfile syntax ကို အသုံးပြု၍ image ၏ dependency များ၊ system library များ၊ နှင့် configuration များကို တိကျစွာ သတ်မှတ်ပေးကြပါသည်-

```
FROM python:3.12
RUN apt-get update
RUN apt-get install -y gcc
RUN apt-get install -y libpq-dev
RUN pip install numpy
RUN pip install pandas
COPY . /app
WORKDIR /app
RUN pip install .
```

အရေးကြီးသော ခြားနားချက်တစ်ခုမှာ- Docker **image** ဆိုသည်မှာ ထုပ်ပိုးထားသော artifact (template တစ်ခုကဲ့သို့) ဖြစ်ပြီး၊ **container** မှာမူ ထို image ၏ မောင်းနှင်နေသော instance တစ်ခု ဖြစ်ပါသည်။ Image တစ်ခုတည်းမှ container အများအပြားကို မောင်းနှင်နိုင်ပါသည်။ Image များကို layer များဖြင့် တည်ဆောက်ထားပြီး Dockerfile ထဲရှိ instruction တိုင်း (`FROM` , `RUN` , `COPY` စသည်) က layer အသစ်တစ်ခုကို ဖန်တီးပါသည်။ Docker သည် ဤ layer များကို cache လုပ်ထားသဖြင့် သင့် Dockerfile ထဲရှိ လိုင်းတစ်ခုကို ပြောင်းလဲပါက ထို layer နှင့် နောက်ဆက်တွဲ layer များကိုသာ ပြန်လည် build လုပ်ရန် လိုအပ်ပါလိမ့်မည်။

ယခင် Dockerfile တွင် ပြဿနာအချို့ ရှိပါသည်- ၎င်းသည် slim variant အစား full Python image ကို အသုံးပြုထားသည်၊ လိုအပ်ချက်ထက် ပိုသော layer များကို ဖန်တီးသည့် သီးခြား `RUN` command များကို မောင်းနှင်ထားသည်၊ ဗားရှင်းများကို pinning ပြုလုပ်ထားခြင်း မရှိပါ၊ ထို့အပြင် မလိုအပ်သော ဖိုင်များကို Shipping ပြုလုပ်ပြီး package manager cache များကို ရှင်းလင်းထားခြင်း မရှိပါ။ အခြား ခဏခဏ မှားတတ်သော အမှားများတွင် container များကို root အဖြစ် မလုံမခြုံ မောင်းနှင်ခြင်း နှင့် မတော်တဆ လျှို့ဝှက်ချက်များ (secrets) ကို layer များအတွင်း မြှုပ်နှံမိခြင်းတို့ ပါဝင်ပါသည်။

အောက်တွင် ပိုမိုကောင်းမွန်အောင် ပြုပြင်ထားသော ဗားရှင်းဖြစ်ပါသည်-

```
FROM python:3.12-slim
COPY --from=ghcr.io/astral-sh/uv:latest /uv /usr/local/bin/uv
RUN apt-get update && \
    apt-get install -y --no-install-recommends gcc libpq-dev && \
    rm -rf /var/lib/apt/lists/*
WORKDIR /app
ENV PATH="/app/.venv/bin:$PATH"
COPY pyproject.toml uv.lock ./
RUN uv sync --locked --no-dev --no-install-project
COPY . .
RUN uv sync --locked --no-dev
```

ယခင်ဥပမာတွင် `uv` ကို source မှ install ပြုလုပ်ခြင်းထက် prebuilt binary ကို `ghcr.io/astral-sh/uv:latest` image ထံမှ ကူးယူထားသည်ကို တွေ့ရပါမည်။ ဤအရာကို *builder* pattern ဟု ခေါ်ဆိုကြပါသည်။ ဤ pattern ဖြင့် ကျွန်ုပ်တို့သည် ကုဒ်ကို compile လုပ်ရန် လိုအပ်သည့် tool များအားလုံးကို Shipping ပြုလုပ်ရန် မလိုဘဲ applicationကို မောင်းနှင်ရန် လိုအပ်သည့် နောက်ဆုံး binary ကိုသာ Shipping ပြုလုပ်ရန် လိုအပ်ပါသည် (ဤနေရာတွင် `uv` ဖြစ်ပါသည်)။

Docker တွင် သတိပြုရမည့် အရေးကြီးသော ကန့်သတ်ချက်များ ရှိပါသည်။ ပထမချက်မှာ container image များသည် မကြာခဏ platform-specific ဖြစ်လေ့ရှိပါသည်— `linux/amd64` အတွက် build ပြုလုပ်ထားသော image သည် နှေးကွေးသော emulation မပါဘဲ `linux/arm64` (Apple Silicon Mac များ) ပေါ်တွင် မူလအတိုင်း အလုပ်လုပ်မည် မဟုတ်ပါ။ ဒုတိယချက်မှာ Docker container များသည် Linux kernel ကို လိုအပ်သဖြင့် macOS နှင့် Windows ပေါ်တွင် Docker သည် နောက်ကွယ်၌ ပေါ့ပါးသော Linux VM တစ်ခုကို မောင်းနှင်ရပြီး overhead ကို တိုးပွားစေပါသည်။ တတိယချက်မှာ Docker ၏ isolation သည် VM များထက် ပိုမို အားနည်းပါသည်—container များသည် host kernel ကို မျှဝေသုံးစွဲကြသဖြင့် multi-tenant environment များတွင် လုံခြုံရေးဆိုင်ရာ စိုးရိမ်စရာ ဖြစ်လာစေပါသည်။

> ယနေ့ခေတ်တွင် ပရောဂျက်အများအပြားသည် **nix flakes** (https://serokell.io/blog/practical-nix-flakes) မှ တစ်ဆင့် ပရောဂျက်တစ်ခုစီအလိုက် “system-wide” librariesနှင့် applicationsကို စီမံခန့်ခွဲရန် nix ကို အသုံးပြုလာကြပါသည်။

စနစ်ပြင်ဆင်မှု (Configuration)

ဆော့ဖ်ဝဲလ်သည် မူလကတည်းက စနစ်ပြင်ဆင်နိုင်စွမ်း (configurable) ရှိပါသည်။ **Command line environment** (https://missing-semester-my.github.io/2026/command-line-environment/) သင်ခန်းစာတွင် ပရိုဂရမ်များသည် flag များ၊ environment variable များ သို့မဟုတ် dotfiles ဟုခေါ်သော configuration ဖိုင်များမှ တစ်ဆင့် option များကို လက်ခံရရှိသည်ကို ကျွန်ုပ်တို့ တွေ့မြင်ခဲ့ရပါသည်။ ဤသည်မှာ ပိုမို ရှုပ်ထွေးသော applicationsအတွက်ပါ မှန်ကန်ပြီး အတိုင်းအတာကြီးမားသော configuration ကို စီမံခန့်ခွဲရန် ခိုင်မာသော pattern များ ရှိကြပါသည်။ ဆော့ဖ်ဝဲလ် configuration ကို ကုဒ်အတွင်း၌ မြုပ်နှံမထားသင့်ဘဲ runtime တွင် ပံ့ပိုးပေးသင့်ပါသည်။ အသုံးများသော နည်းလမ်းအချို့မှာ environment variable များနှင့် config ဖိုင်များ ဖြစ်ကြပါသည်။

အောက်ပါတို့မှာ Environment variable များမှတစ်ဆင့် စနစ်ပြင်ဆင်ထားသော application တစ်ခု၏ ဥပမာ ဖြစ်ပါသည်-

```
import os

DATABASE_URL = os.environ.get("DATABASE_URL", "sqlite:///local.db")
DEBUG = os.environ.get("DEBUG", "false").lower() == "true"
API_KEY = os.environ["API_KEY"]  # လိုအပ်သည် - သတ်မှတ်မထားပါက Error တက်မည်
```

application တစ်ခုကို စနစ်ပြင်ဆင်မှု ဖိုင်တစ်ဆင့်လည်း စနစ်ပြင်ဆင်နိုင်ပါသည် (ဥပမာ- `yaml.load` မှ တစ်ဆင့် config ကို load လုပ်သည့် Python ပရိုဂရမ်)၊ `config.yaml` -

```
database:
  url: "postgresql://localhost/myapp"
  pool_size: 5
server:
  host: "0.0.0.0"
  port: 8080
  debug: false
```

Configuration စနစ်ပြင်ဆင်မှုအကြောင်း စဉ်းစားရာတွင် လက်စွဲလမ်းညွှန်ကောင်းတစ်ခုမှာ ကုဒ်ပြောင်းလဲမှု လုံးဝမပါဘဲ configuration ပြောင်းလဲမှုများဖြင့်သာ တူညီသော codebase ကို မတူညီသော environment များသို့ (development, staging, production) deploy ပြုလုပ်နိုင်ရမည် ဖြစ်ပါသည်။

Configuration စနစ်ပြင်ဆင်မှုများစွာအနက် API key များကဲ့သို့သော အရေးကြီးဒေတာများ (sensitive data) မကြာခဏ ပါဝင်လေ့ရှိပါသည်။ လျှို့ဝှက်ချက်များကို (Secrets) မတော်တဆ ထုတ်ဖော်ပြသမိခြင်းမှ ရှောင်ရှားရန် ဂရုတစိုက် ကိုင်တွယ်ရန် လိုအပ်ပြီး၊ version control အတွင်း၌ ထည့်သွင်းခြင်း မပြုရပါ။

ဝန်ဆောင်မှုများနှင့် စီမံခန့်ခွဲမှု (Services & Orchestration)

ခေတ်သစ် applicationsသည် သီးခြားခွဲထွက်၍ တည်ရှိလေ့ မရှိကြပါ။ သာမန် web application တစ်ခုသည် ဒေတာများ ရေရှည်သိမ်းဆည်းရန်အတွက် database တစ်ခု၊ စွမ်းဆောင်ရည်အတွက် cache တစ်ခု၊ နောက်ကွယ်မှ task များအတွက် message queue တစ်ခု၊ နှင့် အခြား ထောက်ပံ့ပေးသော ဝန်ဆောင်မှုအမျိုးမျိုးကို လိုအပ်နိုင်ပါသည်။ အရာအားလုံးကို monolithic application တစ်ခုတည်းအဖြစ် စုစည်းထားခြင်းထက် ခေတ်သစ် တည်ဆောက်ပုံများသည် လုပ်ဆောင်ချက်များကို သီးခြားစီ ဖွံ့ဖြိုးတိုးတက်စေနိုင်၊ deploy လုပ်နိုင်၊ နှင့် တိုးချဲ့နိုင်သည့် သီးခြား ဝန်ဆောင်မှုများအဖြစ် မကြာခဏ ခွဲထုတ်လေ့ရှိကြပါသည်။

ဥပမာအားဖြင့် ကျွန်ုပ်တို့၏ applicationသည် cache တစ်ခုကို အသုံးပြုခြင်းဖြင့် အကျိုးကျေးဇူး ရရှိနိုင်သည်ဟု ဆုံးဖြတ်ပါက ကိုယ်ပိုင် ဖန်တီးမည့်အစား **Redis** (https://redis.io/) သို့မဟုတ် **Memcached** (https://memcached.org/) ကဲ့သို့သော စမ်းသပ်စစ်ဆေးပြီးသား ခိုင်မာသည့် နည်းလမ်းများကို အသုံးပြုနိုင်ပါသည်။ ကျွန်ုပ်တို့သည် Redis ကို container ၏ အစိတ်အပိုင်းအဖြစ် build ပြုလုပ်၍ application၏ dependency များအတွင်း မြှုပ်နှံထားနိုင်သော်လည်း၊ ထိုသို့ပြုလုပ်ပါက Redis နှင့် ကျွန်ုပ်တို့၏ applicationအကြား dependency အားလုံးကို ညှိနှိုင်းရမည်ဖြစ်ရာ ခက်ခဲနိုင်သလို မဖြစ်နိုင်သည်အထိ ဖြစ်သွားနိုင်ပါသည်။ ထို့အစား ကျွန်ုပ်တို့ ပြုလုပ်နိုင်သည်မှာ application တစ်ခုစီကို ၎င်း၏ သီးခြား container အတွင်း၌ သီးခြားစီ deploy လုပ်ခြင်း ဖြစ်ပါသည်။ ဤအရာကို လေ့ရှိသောအားဖြင့် မူလပါဝင်သည့် အစိတ်အပိုင်းတစ်ခုစီသည် ကွန်ရက်မှတစ်ဆင့် (ပုံမှန်အားဖြင့် HTTP API များမှတစ်ဆင့်) ဆက်သွယ်ပေးသည့် သီးခြား ဝန်ဆောင်မှုအဖြစ် မောင်းနှင်သော microservice architecture ဟု ခေါ်ဆိုကြပါသည်။

Docker Compose (https://docs.docker.com/compose/) သည် container အများအပြားပါဝင်သော applicationsကို သတ်မှတ်ရန်နှင့် မောင်းနှင်ရန်အတွက် tool တစ်ခု ဖြစ်ပါသည်။ Container များကို သီးခြားစီ စီမံခန့်ခွဲခြင်းထက် ဝန်ဆောင်မှုအားလုံးကို YAML ဖိုင်တစ်ခုတည်းတွင် ကြေညာပြီး ၎င်းတို့ကို အတူတကွ စီမံခန့်ခွဲမောင်းနှင် (orchestrate) နိုင်ပါသည်။ ယခုအခါ ကျွန်ုပ်တို့၏ application တစ်ခုလုံးတွင် container တစ်ခုထက်မက ပါဝင်လာပြီ ဖြစ်ပါသည်-

```
# docker-compose.yml
services:
  web:
    build: .
    ports:
      - "8080:8080"
    environment:
      - REDIS_URL=redis://cache:6379
    depends_on:
      - cache

  cache:
    image: redis:7-alpine
    volumes:
      - redis_data:/data

volumes:
  redis_data:
```

`docker compose up` ဖြင့် ဝန်ဆောင်မှု နှစ်ခုစလုံး အတူတကွ စတင်မည်ဖြစ်ပြီး web application သည် hostname `cache` ကို အသုံးပြု၍ Redis ထံသို့ ချိတ်ဆက်နိုင်ပါသည် (Docker ၏ internal DNS က service အမည်များကို အလိုအလျောက် ဖြေရှင်းပေးပါသည်)။ Docker Compose သည် ဝန်ဆောင်မှုတစ်ခုသို့မဟုတ် တစ်ခုထက်မက မည်သို့ deploy ပြုလုပ်လိုသည်ကို ကြေညာစေပြီး၊ ၎င်းတို့ကို အတူတကွ စတင်ခြင်း၊ ၎င်းတို့အကြား networking ပြုလုပ်ပေးခြင်း၊ နှင့် ဒေတာ ရေရှည်တည်တံ့ရေးအတွက် shared volume များကို စီမံခန့်ခွဲခြင်းဆိုင်ရာ orchestration ကို ကိုင်တွယ်ပေးပါသည်။

Production တွင် deploy ပြုလုပ်ရန်အတွက် သင်သည် သင့် docker compose ဝန်ဆောင်မှုများကို စနစ်စတင်သည့်အခါ (boot) အလိုအလျောက် စတင်စေချင်ပြီး ပျက်စီးသွားပါက ပြန်လည် စတင်စေချင်ပါလိမ့်မည်။ အသုံးများသော နည်းလမ်းတစ်ခုမှာ docker compose deployment ကို စီမံခန့်ခွဲရန် systemd ကို အသုံးပြုခြင်း ဖြစ်ပါသည်-

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My Application
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/opt/myapp
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down

[Install]
WantedBy=multi-user.target
```

ဤ systemd unit ဖိုင်သည် သင့်applicationကို စနစ်စတင်ချိန်တွင် (Docker အသင့်ဖြစ်ပြီးနောက်) စတင်ကြောင်း သေချာစေပြီး `systemctl start myapp` ၊ `systemctl stop myapp` ၊ နှင့် `systemctl status myapp` ကဲ့သို့သော စံမိုဘိုင်း/စနစ်ထိန်းချုပ်မှုများကို ပံ့ပိုးပေးပါသည်။

Deployment လိုအပ်ချက်များ ပိုမို ရှုပ်ထွေးလာသည်နှင့်အမျှ—စက်အများအပြားတွင် တိုးချဲ့နိုင်စွမ်း (scalability) လိုအပ်ခြင်း၊ ဝန်ဆောင်မှုများ ပျက်စီးသွားသည့်အခါ fault tolerance ရှိခြင်း၊ နှင့် ရရှိနိုင်မှု မြင့်မားသော (high availability) အာမခံချက်များ လိုအပ်ခြင်း—အဖွဲ့အစည်းများသည် စက်စုစည်းမှုများ (clusters) ပေါ်ရှိ container ထောင်ပေါင်းများစွာကို စီမံခန့်ခွဲနိုင်သည့် Kubernetes (k8s) ကဲ့သို့သော ဆန်းသစ်သော container orchestration platform များကို အသုံးပြုလာကြပါသည်။ သို့သော်လည်း Kubernetes တွင် လေ့လာရန် ခက်ခဲသော သင်ယူမှုမျဉ်းဆွဲနှင့် သုံးစွဲရသည့် စက်လည်ပတ်မှု ကုန်ကျစရိတ် သိသာထင်ရှားစွာ ရှိနေသဖြင့် ပရောဂျက်ငယ်များအတွက် လိုအပ်သည်ထက် ပိုလွန်နေတတ်ပါသည်။

ဤ container အများအပြား ပါဝင်သည့် setup သည် ခေတ်သစ် ဝန်ဆောင်မှုများသည် စံသတ်မှတ်ထားသော API များ (HTTP REST API များ) မှတစ်ဆင့် အပြန်အလှန် ဆက်သွယ်ကြသောကြောင့် အတိုင်းအတာတစ်ခုအထိ ဖြစ်နိုင်ခြင်း ဖြစ်ပါသည်။ ဥပမာအားဖြင့် ပရိုဂရမ်တစ်ခုသည် OpenAI သို့မဟုတ် Anthropic ကဲ့သို့သော LLM provider များနှင့် ဆက်သွယ်တိုင်း၊ နောက်ကွယ်တွင် ၎င်းတို့၏ server များထံ HTTP request ပို့ဆောင်ပြီး ရလဒ်ကို တုံ့ပြန်ချက်စစ်ဆေးခြင်း (parsing) ပြုလုပ်နေခြင်း ဖြစ်ပါသည်။

```
$ curl https://api.anthropic.com/v1/messages \
    -H "x-api-key: $ANTHROPIC_API_KEY" \
    -H "content-type: application/json" \
    -H "anthropic-version: 2023-06-01" \
    -d '{"model": "claude-sonnet-4-20250514", "max_tokens": 256,
         "messages": [{"role": "user", "content": "Explain containers vs VMs in
one sentence."}]}'
```

ထုတ်ဝေခြင်း (Publishing)

သင်၏ ကုဒ်အလုပ်လုပ်ကြောင်း ပြသပြီးသည်နှင့် အခြားသူများ Download ပြုလုပ်၍ install လုပ်နိုင်ရန် အတွက် ၎င်းကို ဖြန့်ဝေပေးရန် စိတ်ဝင်စားပေလိမ့်မည်။ ဖြန့်ဝေခြင်း (Distribution) တွင် ပုံစံအမျိုးမျိုး ရှိပြီး သင်အသုံးပြုသော programming language နှင့် environment တို့နှင့် အစဉ်အမြဲ ဆက်စပ်နေပါသည်။

ဖြန့်ဝေခြင်း၏ အရိုးရှင်းဆုံး ပုံစံမှာ အခြားသူများ မိမိတို့စက်များတွင် Download ပြုလုပ်၍ install ပြုလုပ်နိုင်ရန် artifact များကို တင်ပေးခြင်း (upload ပြုလုပ်ခြင်း) ဖြစ်ပါသည်။ ဤသည်မှာ ယခုထက်ထိ ခေတ်စားနေဆဲ ဖြစ်ပြီး `.deb` ဖိုင်များပါဝင်သည့် HTTP directory listing တစ်ခုဖြစ်သော [Ubuntu ၏ package archive](http://archive.ubuntu.com/ubuntu/pool/main/) ကဲ့သို့သော နေရာများတွင် တွေ့ရှိနိုင်ပါသည်။

ယနေ့ခေတ်တွင် GitHub သည် source code နှင့် artifact များကို ထုတ်ဝေရန်အတွက် de facto platform တစ်ခု ဖြစ်လာခဲ့ပါသည်။ Source code ကို မကြာခဏ အများပြည်သူ လွတ်လပ်စွာ ရယူနိုင်သော်လည်း GitHub Releases သည် ပြုပြင်ထိန်းသိမ်းသူများအား tag တပ်ထားသော ဗားရှင်းများတွင် prebuilt binary များနှင့် အခြား artifact များကို တွဲဆက်ပေးနိုင်စေပါသည်။

Package manager များသည် အချို့အချိန်များတွင် source မှဖြစ်စေ သို့မဟုတ် ကြိုတင် build လုပ်ထားသော wheel မှဖြစ်စေ GitHub မှ တိုက်ရိုက် install ပြုလုပ်ခြင်းကို ထောက်ပံ့ပေးကြပါသည်-

```
# Source မှ install ပြုလုပ်ခြင်း (clone ပြုလုပ်၍ build လုပ်မည်)
$ pip install git+https://github.com/psf/requests.git

# သီးခြား tag/branch မှ install ပြုလုပ်ခြင်း
$ pip install git+https://github.com/psf/requests.git@v2.32.3

# GitHub release မှ wheel ကို တိုက်ရိုက် install ပြုလုပ်ခြင်း
$ pip install https://github.com/user/repo/releases/download/v1.0/package-1.0-p
y3-none-any.whl
```

အမှန်စင်စစ် Go ကဲ့သို့သော အချို့ language များသည် ဗဟိုချုပ်ကိုင်မှုမရှိသော ဖြန့်ဝေရေး မော်ဒယ်ကို အသုံးပြုကြပါသည်—ဗဟို package repository တစ်ခုအဖြစ် မဟုတ်ဘဲ Go module များကို ၎င်းတို့၏ source code repository များထဲမှ တိုက်ရိုက် ဖြန့်ဝေကြပါသည်။ `github.com/gorilla/mux` ကဲ့သို့သော Module လမ်းကြောင်းများသည် ကုဒ်ရှိရာနေရာကို ညွှန်ပြပြီး `go get` က ထိုနေရာထဲမှ တိုက်ရိုက် ဆွဲယူပေးပါသည်။ သို့သော်လည်း `pip` ၊ `cargo` ၊ သို့မဟုတ် `brew` ကဲ့သို့သော package manager အများစုတွင်

ဖြန့်ဝေရလွယ်ကူစေရန်နှင့် install ပြုလုပ်ရလွယ်ကူစေရန်အတွက် ကြိုတင်ထုပ်ပိုးထားသော ပရောဂျက်များ၏ ဗဟို index များ ရှိကြပါသည်။ အကယ်၍ ကျွန်ုပ်တို့သည် အောက်ပါအတိုင်း Run လိုက်ပါက-

```
$ uv pip install requests --verbose --no-cache 2>&1 | grep -F '.whl'
DEBUG Selecting: requests==2.32.5 [compatible] (requests-2.32.5-py3-none-any.wh
l)
DEBUG No cache entry for: https://files.pythonhosted.org/packages/1e/db/4254e3e
abe8020b458f1a747140d32277ec7a271daf1d235b70dc0b4e6e3/requests-2.32.5-py3-none-
any.whl.metadata
DEBUG No cache entry for: https://files.pythonhosted.org/packages/1e/db/4254e3e
abe8020b458f1a747140d32277ec7a271daf1d235b70dc0b4e6e3/requests-2.32.5-py3-none-
any.whl
```

ကျွန်ုပ်တို့သည် `requests` wheel ကို မည်သည့်နေရာမှ ဆွဲယူနေသည်ဆိုသည်ကို မြင်တွေ့နိုင်ပါသည်။ ဖိုင်အမည်ရှိ `py3-none-any` ကို သတိပြုပါ—၎င်း၏ အဓိပ္ပာယ်မှာ ဤ wheel သည် မည်သည့် OS၊ မည်သည့် architecture ပေါ်တွင်မဆို မည်သည့် Python 3 ဗားရှင်းဖြင့်မဆို အလုပ်လုပ်သည်ဟု ဆိုလိုခြင်း ဖြစ်ပါသည်။ Compile လုပ်ထားသော ကုဒ်ပါဝင်သည့် package များအတွက် wheel သည် platform အလိုက် သီးခြား ဖြစ်ပါသည်-

```
$ uv pip install numpy --verbose --no-cache 2>&1 | grep -F '.whl'
DEBUG Selecting: numpy==2.2.1 [compatible] (numpy-2.2.1-cp312-cp312-macosx_14_0
_arm64.whl)
```

ဤနေရာတွင် `cp312-cp312-macosx_14_0_arm64` က ဤ wheel သည် ARM64 (Apple Silicon) အတွက် macOS 14+ ပေါ်ရှိ CPython 3.12 အတွက် သီးသန့် ဖြစ်ကြောင်း ညွှန်ပြနေပါသည်။ အကယ်၍ သင်သည် အခြား platform တစ်ခုပေါ်တွင် ရောက်ရှိနေပါက `pip` သည် မတူညီသော wheel တစ်ခုကို Download ပြုလုပ်မည် သို့မဟုတ် source မှ build ပြုလုပ်ပါလိမ့်မည်။

ဆန့်ကျင်ဘက်အားဖြင့် ကျွန်ုပ်တို့ ဖန်တီးခဲ့သော package ကို အခြားသူများ ရှာဖွေတွေ့ရှိနိုင်ရန်အတွက် ကျွန်ုပ်တို့သည် ၎င်းကို ဤ registry များအနက် တစ်ခုခုထံ ထုတ်ဝေ (publish) ရန် လိုအပ်ပါသည်။ Python တွင် ပင်မ registry မှာ **Python Package Index (PyPI)** (https://pypi.org) ဖြစ်ပါသည်။ Install ပြုလုပ်ခြင်းကဲ့သို့ပင် package များကို ထုတ်ဝေရန် နည်းလမ်းများစွာ ရှိပါသည်။ `uv publish` command သည် package များကို PyPI ထံ upload လုပ်ရန်အတွက် ခေတ်မီသော interface တစ်ခုကို ပံ့ပိုးပေးပါသည်-

```
$ uv publish --publish-url https://test.pypi.org/legacy/
Publishing greeting-0.1.0.tar.gz
Publishing greeting-0.1.0-py3-none-any.whl
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် တကယ့် PyPI ကို မည်စည်မ်းစေဘဲ သင်၏ ထုတ်ဝေရေး workflow ကို စမ်းသပ်ရန် ရည်ရွယ်ထားသော သီးခြား package registry တစ်ခု ဖြစ်သည့် **TestPyPI** (https://test.pypi.org) ကို အသုံးပြုနေကြပါသည်။ Upload လုပ်ပြီးပါက TestPyPI ထံမှ install ပြုလုပ်နိုင်ပြီ ဖြစ်ပါသည်-

```
$ uv pip install --index-url https://test.pypi.org/simple/ greeting
```

ဆော့ဖ်ဝဲလ် ထုတ်ဝေရာတွင် အဓိက စဉ်းစားရမည့် အချက်မှာ ယုံကြည်စိတ်ချရမှု (trust) ဖြစ်ပါသည်။ သုံးစွဲသူများသည် ၎င်းတို့ Download ပြုလုပ်သော package သည် သင့်ထံမှ အမှန်တကယ် လာခြင်းဖြစ်ကြောင်း နှင့် ပြုပြင်မွမ်းမံထားခြင်း မရှိကြောင်း မည်သို့ အတည်ပြုကြမည်နည်း။ Package registry များသည် တည်ငြိမ်မှုကန်မှုကို အတည်ပြုရန် checksum များကို အသုံးပြုကြပြီး အချို့သော ecosystem များသည် မူပိုင်ခွင့်ဆိုင်ရာ သက်သေတပြ အထောက်အထား ပံ့ပိုးရန် package လက်မှတ်ရေးထိုးခြင်း (package signing) ကို ထောက်ပံ့ပေးကြပါသည်။

မတူညီသော language များတွင် မိမိတို့၏ ကိုယ်ပိုင် package registry များ ရှိကြပါသည်- Rust အတွက် **crates.io** (https://crates.io)၊ JavaScript အတွက် **npm** (https://www.npmjs.com)၊ Ruby အတွက် **RubyGems** (https://rubygems.org)၊ နှင့် container image များအတွက် **Docker Hub** (https://hub.docker.com) ဖြစ်ကြပါသည်။ ထိုအတောအတွင်း သီးသန့် သို့မဟုတ် အတွင်းပိုင်း package များအတွက် အဖွဲ့အစည်းများသည် မိမိတို့၏ ကိုယ်ပိုင် package repository များကို မကြာခဏ deploy လုပ်လေ့ ရှိကြပါသည် (သီးသန့် PyPI server သို့မဟုတ် သီးသန့် Docker registry ကဲ့သို့သော) သို့မဟုတ် cloud provider များထံမှ စီမံခန့်ခွဲပေးထားသော နည်းလမ်းများကို အသုံးပြုကြပါသည်။

ဝဘ်ဝန်ဆောင်မှုတစ်ခုကို အင်တာနက်ထံ deploy ပြုလုပ်ခြင်းတွင် အပိုဆောင်း အဆောက်အအုံများ ပါဝင်ပါသည်- domain name မှတ်ပုံတင်ခြင်း၊ သင့် domain ကို သင့် server ထံ ညွှန်းဆိုရန် DNS စနစ်ပြင်ဆင်ခြင်း၊ နှင့် မကြာခဏဆိုသလို HTTPS ကို ကိုင်တွယ်ရန်နှင့် traffic လမ်းကြောင်းပေးရန် nginx ကဲ့သို့သော reverse proxy တစ်ခု သုံးစွဲခြင်း။ Documentation များ သို့မဟုတ် static site များကဲ့သို့ ပိုမို ရိုးရှင်းသော အသုံးပြုမှုများအတွက် **GitHub Pages** (https://pages.github.com/) သည် repository ထံမှ တိုက်ရိုက် အခမဲ့ hosting ကို ပံ့ပိုးပေးပါသည်။

လေ့ကျင့်ခန်းများ (Exercises)

1. သင့် environment ကို `printenv` ဖြင့် ဖိုင်တစ်ခုထဲသို့ သိမ်းဆည်းပါ၊ venv တစ်ခု ဖန်တီးပါ၊ ၎င်းကို activate ပြုလုပ်ပါ၊ အခြားဖိုင်တစ်ခုထဲသို့ `printenv` လုပ်ပြီး `diff before.txt after.txt` ပြုလုပ်ပါ။ Environment တွင် မည်သည့်အရာများ ပြောင်းလဲသွားသနည်း။ Shell က venv ကို အဘယ်ကြောင့် ဦးစားပေးသနည်း။ (အချက်အလက်- activation ပြုလုပ်မှုနှင့် ပြုလုပ်ပြီးနောက် `$PATH` ကို ကြည့်ပါ။) `which deactivate` ကို မောင်းနှင်ပြီး deactivate bash function သည် မည်သည့်အရာ ပြုလုပ်နေသည်ကို အကြောင်းပြချက် ဖော်ထုတ်ပါ။
2. `pyproject.toml` ပါရှိသော Python package တစ်ခုကို ဖန်တီးပြီး ၎င်းကို virtual environment တစ်ခုတွင် install ပြုလုပ်ပါ။ Lockfile တစ်ခု ဖန်တီးပြီး စစ်ဆေးပါ။
3. Docker ကို install ပြုလုပ်ပြီး docker compose ကို အသုံးပြု၍ Missing Semester သင်တန်း ဝဘ်ဆိုက်ကို မိမိစက်တွင် build ပြုလုပ်ရန် ၎င်းကို အသုံးပြုပါ။
4. ရိုးရှင်းသော Python application တစ်ခုအတွက် Dockerfile တစ်ခု ရေးသားပါ။ ထို့နောက် Redis cache နှင့်အတူ သင့် application ကို မောင်းနှင်သည့် `docker-compose.yml` တစ်ခု ရေးသားပါ။
5. Python package တစ်ခုကို TestPyPI ထံ ထုတ်ဝေပါ (ဝေမျှထိုက်ပါမှလွဲ၍ တကယ့် PyPI ထံ ထုတ်ဝေခြင်း မပြုပါနှင့်!)။ ထို့နောက် ထို package ဖြင့် Docker image တစ်ခုကို build ပြုလုပ်ပြီး ၎င်းကို `ghcr.io` ထံ push ပြုလုပ်ပါ။
6. **GitHub Pages** (https://docs.github.com/en/pages/quickstart) ကို အသုံးပြု၍ ဝဘ်ဆိုက်တစ်ခု ပြုလုပ်ပါ။ Extra (non-)credit- ၎င်းကို custom domain တစ်ခုဖြင့် စနစ်ပြင်ဆင်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. PyPI	https://pypi.org/	
2. `uv`	https://docs.astral.sh/uv/	
3. `hatch`	https://hatch.pypa.io/	
4. `poetry`	https://python-poetry.org/	
5. PEP 517	https://peps.python.org/pep-0517/	
6. PEP 621	https://peps.python.org/pep-0621/	
7. ripgrep ၏ releases စာမျက်နှာ	https://github.com/BurntSushi/ripgrep/releases	
8. Semantic Versioning	https://semver.org/	
9. Nix	https://nixos.org/	
10. Bazel	https://bazel.build/	
11. Code Quality and CI	https://missing-semester-my.github.io/2026/code-quality/	
12. Dependabot	https://github.com/dependabot	
13. Docker	https://www.docker.com/	
14. nix flakes	https://serokell.io/blog/practical-nix-flakes	
15. Command line environment	https://missing-semester-my.github.io/2026/command-line-environment/	

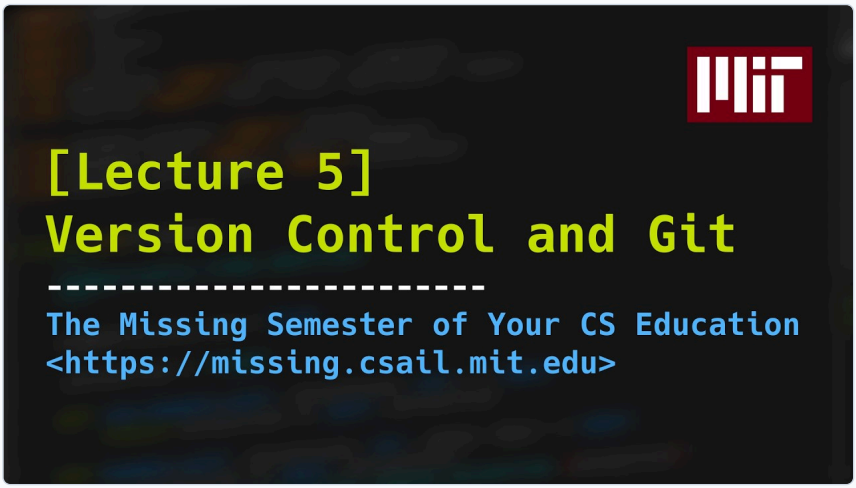
Resource / Description	URL	Micro QR
16. Redis	https://redis.io/	
17. Memcached	https://memcached.org/	
18. Docker Compose	https://docs.docker.com/compose/	
19. Ubuntu ၏ package archive	http://archive.ubuntu.com/ubuntu/pool/main/	
20. Python Package Index (PyPI)	https://pypi.org	
21. TestPyPI	https://test.pypi.org	
22. crates.io	https://crates.io	
23. npm	https://www.npmjs.com	
24. RubyGems	https://rubygems.org	
25. Docker Hub	https://hub.docker.com	
26. GitHub Pages	https://pages.github.com/	
27. Sphinx	https://www.sphinx-doc.org/	
28. MkDocs	https://www.mkdocs.org/	
29. Read the Docs	https://readthedocs.org/	
30. OpenAPI specification	https://www.openapis.org/	
31. GitHub Pages	https://docs.github.com/en/pages/quickstart	

Version Control နှင့် Git

အနှစ်ချုပ် - Git ၏ data model အကြောင်းနှင့် Version control ပြုလုပ်ခြင်း၊ ပူးပေါင်းဆောင်ရွက်ခြင်းတို့အတွက် Git ကို မည်သို့ အသုံးပြုရမည်ကို လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Version Control နှင့် Git



Scan the QR code below using your mobile camera or click the link to watch this full lecture online:

Direct Online Link:

https://www.youtube.com/watch?v=9K8IB61dI3Y

Version control systems (VCSs) ဆိုသည်မှာ source code များ (သို့မဟုတ် အခြား ဖိုင်နှင့် ဖိုဒါ စုစည်းမှုများ) ၏ ပြောင်းလဲမှုများကို စောင့်ကြည့်မှတ်တမ်းတင်ရန် အသုံးပြုသော tools များ ဖြစ်ကြသည်။ အမည်တွင် ပါရှိသည့်အတိုင်း ဤ tools များသည် ပြောင်းလဲမှု မှတ်တမ်း (history) ကို ထိန်းသိမ်းထားနိုင်ရန် ကူညီပေးရုံမျှမက ပူးပေါင်းဆောင်ရွက်မှုကိုလည်း ပိုမိုလွယ်ကူ စေသည်။ လောဂျစ်အားဖြင့် VCS များသည် ဖိုဒါတစ်ခုနှင့် ၎င်း၏ ပါဝင်သည့်အရာများ၏ ပြောင်းလဲမှုများကို *snapshots* စီးရီးအဖြစ် စောင့်ကြည့် မှတ်တမ်းတင်ပေးပြီး၊ snapshot တစ်ခုစီသည် ပင်မ directory အတွင်းရှိ ဖိုင်များ/ဖိုဒါများ၏ စုစုပေါင်း အခြေအနေကို ပေါင်းစည်း သိမ်းဆည်းထားသည်။ ထို့အပြင် VCS များသည် snapshot တစ်ခုစီကို မည်သူ ဖန်တီးခဲ့သည်၊ snapshot တစ်ခုစီနှင့် သက်ဆိုင်သည့် မက်ဆေ့ဂျ်များ စသည့် metadata များကိုလည်း ထိန်းသိမ်းထားပေးသည်။

Version control သည် အဘယ်ကြောင့် အသုံးဝင်သနည်း။ သင်တစ်ဦးတည်း အလုပ်လုပ်နေချိန်တွင်ပင် ပရောဂျက်၏ ယခင် snapshot များကို ပြန်ကြည့်နိုင်ခြင်း၊ အချို့သော ပြောင်းလဲမှုများကို အဘယ်ကြောင့် ပြုလုပ်ခဲ့သည်ကို မှတ်တမ်းတင်ထားနိုင်ခြင်း၊ ပြီးတူ ဖွံ့ဖြိုးတိုးတက်ရေး branch များတွင် အလုပ်လုပ်နိုင်ခြင်းနှင့် အခြားအရာများစွာကို ပြုလုပ်နိုင်စေသည်။ အခြားသူများနှင့် ပူးပေါင်းလုပ်ဆောင်ရာတွင် အခြားသူများ မည်သည့်အရာများ ပြောင်းလဲထားသည်ကို ကြည့်ရှုရန်နှင့် ပြီးတူ လုပ်ဆောင်ရာတွင် ဖြစ်ပေါ်လာသည့် ပဋိပက္ခများ (conflicts) ကို ဖြေရှင်းရန်အတွက် အလွန်တန်ဖိုးရှိသော tool တစ်ခု ဖြစ်သည်။

ခေတ်မီ VCS များသည် အောက်ပါမေးခွန်းများကို လွယ်ကူစွာ (မကြာခဏဆိုသလို အလိုအလျောက်) ဖြေကြားနိုင်စေသည် -

- ဤ module ကို မည်သူရေးသားခဲ့သနည်း။
- ဤဖိုင်၏ ဤလိုင်းသီးသန့်ကို မည်သည့်အချိန်တွင် ပြင်ဆင်ခဲ့သနည်း။ မည်သူက ပြင်ဆင်ခဲ့သနည်း။ အဘယ်ကြောင့် ပြင်ဆင်ခဲ့သနည်း။
- လွန်ခဲ့သည့် revision ၁၀၀၀ ၏ ကာလအတွင်း၊ သီးသန့် unit test တစ်ခုသည် မည်သည့်အချိန်တွင်/အဘယ်ကြောင့် အလုပ်မလုပ်တော့ဘဲ ရပ်တန့်သွားသနည်း။

အခြား VCS များလည်း ရှိသော်လည်း **Git** သည် version control အတွက် လက်တွေ့ကျကျ အဓိက စံနှုန်းတစ်ခု (de facto standard) ဖြစ်သည်။ ဤ **XKCD comic** (https://xkcd.com/1597/) တွင် Git ၏ ကျော်ကြားမှုကို သရုပ်ဖော်ထားသည် -

!**xkcd 1597** (https://imgs.xkcd.com/comics/git.png)

Git ၏ interface သည် leaky abstraction တစ်ခုဖြစ်သောကြောင့် Git ကို အထက်မှ အောက်သို့ (၎င်း၏ interface / command-line interface မှ စတင်၍) လေ့လာခြင်းသည် ရှုပ်ထွေးမှုများစွာကို ဖြစ်ပေါ်စေနိုင်သည်။ command အနည်းငယ်ကို အလွတ်ကျက်မှတ်ပြီး ဂါထာမန္တန်များကဲ့သို့ မှတ်ယူကာ တစ်ခုခု အမှားအယွင်းဖြစ်သည့်အခါတိုင်း အထက်ပါ ကာတွန်းထဲမှ နည်းလမ်းအတိုင်း လိုက်လုပ်နေမိနိုင်သည်။

Git တွင် ရုပ်ဆိုးသော interface တစ်ခု ရှိသည်ကို ဝန်ခံရမည်ဖြစ်သော်လည်း ၎င်း၏ နောက်ကွယ်မှ ဒီဇိုင်းနှင့် စိတ်ကူးများမှာ အလွန်လှပပါသည်။ ရုပ်ဆိုးသော interface ကို *အလွတ်ကျက်မှတ်* ရမည် ဖြစ်သော်လည်း၊ လှပသော ဒီဇိုင်းကိုမူ *နားလည် သဘောပေါက်* နိုင်ပါသည်။ ထို့ကြောင့် ကျွန်ုပ်တို့သည် Git ကို ၎င်း၏ data model မှ စတင်၍ အောက်မှ အထက်သို့ ရှင်းပြမည်ဖြစ်ပြီး နောက်ပိုင်းတွင်မှ command-line interface ကို လွှမ်းခြုံ ရှင်းပြသွားမည် ဖြစ်သည်။ Data model ကို နားလည်သွားပါက command များသည် အခြေခံ data model ကို မည်သို့ ပြုပြင်ပြောင်းလဲသည် ဆိုသည်ကို ပိုမိုကောင်းမွန်စွာ နားလည်နိုင်မည် ဖြစ်သည်။

Git ၏ data model

Git ၏ တီထွင်ဖန်တီးနိုင်စွမ်းသည် မှတ်တမ်း ထိန်းသိမ်းခြင်း၊ branch များကို ပံ့ပိုးပေးခြင်းနှင့် ပူးပေါင်းဆောင်ရွက်မှုကို ဖြစ်စေခြင်းကဲ့သို့သော version control ၏ ကောင်းမွန်သည့် စွမ်းဆောင်ရည် အားလုံးကို ဖြစ်ပေါ်စေသည့် သေချာစွာ စဉ်းစားထားသော ၎င်း၏ data model တွင် ရှိသည်။

Snapshots

Git သည် ပင်မ directory အတွင်းရှိ ဖိုင်များနှင့် ဖိုဒါများ စုစည်းမှု၏ မှတ်တမ်းကို snapshot စီးရီးတစ်ခုအဖြစ် ပုံစံထုတ် (model) ထားသည်။ Git ဝေါဟာရတွင် ဖိုင်တစ်ခုကို “blob” ဟု ခေါ်ပြီး ၎င်းသည် byte အစုအဝေးတစ်ခုမျှသာ ဖြစ်သည်။ Directory တစ်ခုကို “tree” ဟု ခေါ်ပြီး ၎င်းသည် အမည်များကို blobs သို့မဟုတ် trees များနှင့် ချိတ်ဆက်ပေးသည် (ထို့ကြောင့် directory များတွင် အခြား directory များ ပါဝင်နိုင်သည်)။ Snapshot သည် စောင့်ကြည့် စောင့်ရှောက်ခံနေရသည့် ထိပ်ဆုံးအဆင့် tree ဖြစ်သည်။ ဥပမာအားဖြင့် ကျွန်ုပ်တို့တွင် အောက်ပါအတိုင်း tree တစ်ခု ရှိနိုင်သည်။

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

ထိပ်ဆုံးအဆင့် tree တွင် element နှစ်ခု ပါဝင်သည် - tree “foo” (၎င်းကိုယ်တိုင်၌ element တစ်ခုဖြစ်သော blob “bar.txt” ပါဝင်သည်) နှင့် blob “baz.txt” တို့ ဖြစ်ကြသည်။

မှတ်တမ်းကို ပုံစံထုတ်ခြင်း - Snapshots များကို ဆက်စပ်ခြင်း

Version control system တစ်ခုသည် snapshot များကို မည်သို့ ဆက်စပ်သင့်သနည်း။ ရိုးရှင်းသော ပုံစံတစ်ခုမှာ မျဉ်းဖြောင့်အတိုင်း သွားသော မှတ်တမ်း (linear history) ရှိခြင်း ဖြစ်သည်။ မှတ်တမ်းတစ်ခုသည် အချိန်အလိုက် စီထားသော snapshot များ၏ စာရင်းတစ်ခု ဖြစ်လိမ့်မည်။ အကြောင်းပြချက်များစွာကြောင့် Git သည် ဤကဲ့သို့ ရိုးရှင်းသော ပုံစံကို မသုံးပါ။

Git တွင် မှတ်တမ်းတစ်ခုသည် snapshots များ၏ directed acyclic graph (DAG) တစ်ခု ဖြစ်သည်။ ထိုစကားရပ်သည် ခန်းနားသော သချိၤဝေါဟာရတစ်ခုကဲ့သို့ ထင်ရနိုင်သော်လည်း မကြောက်ပါနှင့်။ ဤအရာ၏ အဓိပ္ပာယ်မှာ Git ရှိ snapshot တိုင်းသည် ၎င်း၏ ရှေ့တွင် ရှိခဲ့သော snapshot များ ဖြစ်သည့် “parents” အစုအဝေးကို ညွှန်းဆိုနေခြင်း ဖြစ်သည်။ snapshot တစ်ခုသည် ပြိုင်တူ လုပ်ဆောင်နေသော ဖွံ့ဖြိုးတိုးတက်ရေး branch နှစ်ခုကို ပေါင်းစည်းခြင်း (merging) ကဲ့သို့သော အကြောင်းရင်းများကြောင့် parent အများအပြားမှ ဆင်းသက်လာနိုင်သောကြောင့် (linear history တွင် ဖြစ်မည်ကဲ့သို့) parent တစ်ခုတည်း မဟုတ်ဘဲ parent အစုအဝေး ဖြစ်နေခြင်း ဖြစ်သည်။

Git သည် ဤ snapshots များကို “commit” များဟု ခေါ်သည်။ Commit history တစ်ခုကို အမြင်ပုံဖော်ကြည့်ပါက အောက်ပါအတိုင်း တွေ့ရပါလိမ့်မည် -

```
o <-- o <-- o <-- o
            ^
             \
              --- o <-- o
```

အထက်ပါ ASCII art တွင် `o` များသည် သီးခြား commit (snapshot) များကို ကိုယ်စားပြုသည်။ မြား များသည် commit တစ်ခုစီ၏ parent ကို ညွှန်ပြသည် (“နောက်မှလာသည်” ဆက်ဆံရေး မဟုတ်ဘဲ “ရှေ့မှလာသည်” ဆက်ဆံရေး ဖြစ်သည်)။ တတိယမြောက် commit ပြီးနောက် မှတ်တမ်းသည် သီးခြား branch နှစ်ခုအဖြစ် ခွဲထွက်သွားသည်။ ၎င်းသည် ဥပမာအားဖြင့် သီးခြား feature နှစ်ခုကို တစ်ခုနှင့်တစ်ခု သီးခြားစီ ပြိုင်တူ ဖွံ့ဖြိုးတိုးတက်စေခြင်း ဖြစ်နိုင်သည်။ နောင်တွင် ဤ branch များကို ပေါင်းစည်းပြီး feature နှစ်ခုလုံး ပါဝင်သည့် snapshot သစ်တစ်ခု ဖန်တီးနိုင်သည်၊ ထိုအခါ အသစ်ဖန်တီးလိုက်သော merge commit ကို စာလုံးမည်းဖြင့် ပြသထားသည့် အောက်ပါအတိုင်း မှတ်တမ်းသစ်တစ်ခု ဖြစ်ပေါ်လာမည် -

```
o <-- o <-- o <-- o <---- o
```bash
        ^            /
         \          v
          --- o <-- o
```
```

Git ရှိ Commit များသည် ပြောင်းလဲ၍မရပါ (immutable)။ သို့သော် ဤသည်မှာ အမှားများကို ပြင်ဆင်၍ မရနိုင်ဟု အဓိပ္ပာယ်မရပါ၊ commit history ကို “ပြင်ဆင်ခြင်း” ဆိုသည်မှာ အမှန်တကယ်တွင် စာမျက်နှာသစ် commit အသစ်များကို ဖန်တီးလိုက်ခြင်းဖြစ်ပြီး ညွှန်းဆိုချက်များ (references - အောက်တွင် ကြည့်ပါ) ကို commit အသစ်များသို့ ညွှန်ပြရန် အပ်ဒိတ်လုပ်လိုက်ခြင်း ဖြစ်သည်။

Data model ကို Pseudocode အဖြစ် ကြည့်ခြင်း

Git ၏ data model ကို pseudocode ဖြင့် ရေးသားထားသည်ကို ကြည့်ရှုခြင်းသည် သင်ယူရန် အထောက်အကူဖြစ်စေနိုင်ပါသည် -

```
// a file is a bunch of bytes
type blob = array<byte>

// a directory contains named files and directories
type tree = map<string, tree | blob>

// a commit has parents, metadata, and the top-level tree
type commit = struct {
    parents: array<commit>
    author: string
    message: string
    snapshot: tree
}
```

၎င်းသည် သပ်ရပ်ပြီး ရိုးရှင်းသော မှတ်တမ်း ပုံစံတစ်ခု ဖြစ်သည်။

Objects နှင့် content-addressing

“object” ဆိုသည်မှာ blob၊ tree သို့မဟုတ် commit ဖြစ်သည် -

```
type object = blob | tree | commit
```

Git ၏ data store တွင် object အားလုံးကို ၎င်းတို့၏ **SHA-1 hash** (https://en.wikipedia.org/wiki/SHA-1) ဖြင့် content-addressed လုပ်ထားသည်။

```
objects = map<string, object>

def store(object):
    id = sha1(object)
    objects[id] = object

def load(id):
    return objects[id]
```

Blobs၊ trees နှင့် commits များကို ဤနည်းဖြင့် ပေါင်းစည်းထားသည် - ၎င်းတို့အားလုံးသည် objects များဖြစ်ကြသည်။ ၎င်းတို့သည် အခြား object များကို ညွှန်းဆိုသည့်အခါ disk ပေါ်ရှိ ၎င်းတို့၏ ကိုယ်စားပြုမှုတွင် အမှန်တကယ် *ပါဝင်နေခြင်း* မဟုတ်ဘဲ ၎င်းတို့၏ hash ဖြင့် ညွှန်းဆိုချက်သာ ရှိကြသည်။

ဥပမာအားဖြင့် [အထက်ပါ](#) ဥပမာ directory ဖွဲ့စည်းပုံအတွက် tree သည် (`git cat-file -p 698281bc680d1995c5f4caaf3359721a5a58d48d` ကို အသုံးပြု၍ ကြည့်ရှုထားသည်) အောက်ပါအတိုင်း ဖြစ်သည် -

```
100644 blob 4448adbf7ecd394f42ae135bbeed9676e894af85    baz.txt
040000 tree c68d233a33c5c06e0340e4c224f0afca87c8ce87    foo
```

Tree ကိုယ်တိုင်တွင် ၎င်း၏ ပါဝင်သည့်အရာများဖြစ်သော `baz.txt` (blob) နှင့် `foo` (tree) သို့ ညွှန်ပြသည့် pointer များ ပါဝင်သည်။ `git cat-file -p 4448adbf7ecd394f42ae135bbeed9676e894af85` ဖြင့် baz.txt နှင့် သက်ဆိုင်သော hash ဖြင့် ညွှန်းဆိုထားသည့် အကြောင်းအရာကို ကြည့်ရှုပါက အောက်ပါအတိုင်း ရရှိမည်ဖြစ်သည် -

```
git is wonderful
```

References (ညွှန်းဆိုချက်များ)

ယခုအခါ snapshot အားလုံးကို ၎င်းတို့၏ SHA-1 hash များနှင့် ခွဲခြားသတ်မှတ်နိုင်ပြီ ဖြစ်သည်။ သို့သော် လူများသည် ၁၆ ခြောက်လီစနစ် (hexadecimal) စာလုံး ၄၀ ပါသော စာကြောင်းများကို မှတ်မိရန် မလွယ်ကူသောကြောင့် ၎င်းမှာ အဆင်မပြေပါ။

ဤပြဿနာအတွက် Git ၏ ဖြေရှင်းချက်မှာ “references” ဟုခေါ်သော SHA-1 hash များအတွက် လူများဖတ်ရှုနိုင်သည့် အမည်များ ဖြစ်သည်။ References များသည် commit များကို ညွှန်ပြသော pointer များ ဖြစ်ကြသည်။ ပြောင်းလဲ၍မရသော (immutable) object များနှင့် မတူဘဲ references များသည် ပြောင်းလဲနိုင်သည် (mutable - commit အသစ်ကို ညွှန်ပြရန် အပ်ဒိတ်လုပ်နိုင်သည်)။ ဥပမာအားဖြင့် `master` reference သည် များသောအားဖြင့် ပင်မ ဖွံ့ဖြိုးတိုးတက်ရေး branch ၏ နောက်ဆုံး commit ကို ညွှန်ပြလေ့ရှိသည်။

```
references = map<string, string>

def update_reference(name, id):
    references[name] = id

def read_reference(name):
    return references[name]

def load_reference(name_or_id):
    if name_or_id in references:
        return load(references[name_or_id])
    else:
        return load(name_or_id)
```

ဤအရာဖြင့် Git သည် ရှည်လျားသော hexadecimal စာကြောင်းအစား မှတ်တမ်းရှိ သီးခြား snapshot တစ်ခု ကို ညွှန်းဆိုရန် “master” ကဲ့သို့သော လူများ ဖတ်ရှုနိုင်သည့် အမည်များကို အသုံးပြုနိုင်သည်။

အသေးစိတ်တစ်ခုမှာ snapshot အသစ်တစ်ခု ရယူသည့်အခါ မည်သည့်အရာနှင့် နှိုင်းယှဉ်ထားသည်ကို သိရှိ စေရန် (commit ၏ `parents` field ကို မည်သို့ သတ်မှတ်ရမည်ဆိုသည်) မှတ်တမ်းတွင် “ကျွန်ုပ်တို့ လက်ရှိ မည်သည့်နေရာသို့ ရောက်ရှိနေသည်” ဆိုသည့် အယူအဆကို လိုချင်ကြသည်။ Git တွင် ထို “ကျွန်ုပ်တို့ လက်ရှိ ရောက်ရှိနေသည့်နေရာ” သည် “HEAD” ဟုခေါ်သော အထူး reference တစ်ခု ဖြစ်သည်။

Repositories

နောက်ဆုံးတွင် Git *repository* ၏ အဓိပ္ပာယ်ကို (ကြမ်းဖျင်းအားဖြင့်) အဓိပ္ပာယ်ဖွင့်ဆိုနိုင်သည် - ၎င်းသည် data `objects` နှင့် `references` တို့ ဖြစ်သည်။

Disk ပေါ်တွင် Git သိမ်းဆည်းထားသမျှ အရာအားလုံးသည် objects နှင့် references များဖြစ်သည် - ၎င်း သည် Git ၏ data model တွင် ရှိသမျှ အရာအားလုံးပင် ဖြစ်သည်။ `git` command အားလုံးသည် objects များကို ထည့်သွင်းခြင်းနှင့် references များကို ထည့်သွင်းခြင်း/အပ်ဒိတ်လုပ်ခြင်းဖြင့် commit DAG ကို ပြုပြင်ပြောင်းလဲမှု အချို့ ပြုလုပ်ခြင်းသို့ ချိတ်ဆက်နေသည်။

Command တစ်ခုခုကို ရိုက်ထည့်သည့်အခါတိုင်း၊ ၎င်း command သည် အခြေခံ graph data structure ကို မည်သည့် ပြုပြင်ပြောင်းလဲမှု ပြုလုပ်နေသည်ဆိုသည်ကို စဉ်းစားပါ။ အပြန်အလှန်အားဖြင့် commit DAG ကို “commit မလုပ်ရသေးသော ပြောင်းလဲမှုများကို ပယ်ဖျက်ပြီး ‘master’ ref ကို commit `5d83f9e` သို့ ညွှန်ပြ စေခြင်း” ကဲ့သို့သော သီးခြား ပြောင်းလဲမှုမျိုး ပြုလုပ်ရန် ကြိုးစားနေပါက ၎င်းကို ပြုလုပ်ရန် command တစ်ခုခု ရှိနိုင်ပါသည် (ဥပမာ - ဤဖြစ်ရပ်တွင် `git checkout master; git reset --hard 5d83f9e`)။

Staging area

ဤသည်မှာ data model နှင့် သီးခြားစီ ဖြစ်သော်လည်း commit များကို ဖန်တီးရန် interface ၏ အစိတ်အပိုင်း တစ်ခု ဖြစ်သော အခြား အယူအဆတစ်ခု ဖြစ်သည်။

အထက်တွင် ဖော်ပြခဲ့သည့်အတိုင်း snapshot ရယူခြင်းကို အကောင်အထည်ဖော်ရန် စဉ်းစားနိုင်သည့် နည်းလမ်းတစ်ခုမှာ working directory ၏ *လက်ရှိ အခြေအနေ* ပေါ် အခြေခံ၍ snapshot အသစ်တစ်ခု ဖန်တီးပေးသည့် "create snapshot" command တစ်ခု ရှိခြင်း ဖြစ်သည်။ အချို့သော version control tools များသည် ဤကဲ့သို့ အလုပ်လုပ်သော်လည်း Git ကမူ မဟုတ်ပါ။ ကျွန်ုပ်တို့သည် သပ်ရပ်သန့်ရှင်းသော snapshot များကို လိုချင်ကြပြီး လက်ရှိ အခြေအနေမှ snapshot ပြုလုပ်ခြင်းသည် အမြဲတမ်း သင့်တော်မည် မဟုတ်ပါ။ ဥပမာ အားဖြင့် သင်သည် သီးခြား feature နှစ်ခုကို အကောင်အထည်ဖော်ထားပြီး သီးခြား commit နှစ်ခု ဖန်တီးလိုသည့် အခြေအနေကို မြင်ယောင်ကြည့်ပါ၊ ပထမတစ်ခုက ပထမ feature ကို မိတ်ဆက်ပေးပြီး နောက်တစ်ခုက ဒုတိယ feature ကို မိတ်ဆက်ပေးမည် ဖြစ်သည်။ သို့မဟုတ် အမှားပြင်ဆင်မှု (bugfix) တစ်ခုနှင့်အတူ သင့် ကုဒ်တစ်လျှောက် ထည့်သွင်းထားသော debugging print စာကြောင်းများ ရှိနေသည့် အခြေအနေကို မြင်ယောင်ကြည့်ပါ၊ print စာကြောင်းများအားလုံးကို ပယ်ဖျက်ပြီး bugfix ကိုသာ commit လုပ်လိုမည် ဖြစ်သည်။

Git သည် "staging area" ဟုခေါ်သော နည်းလမ်းမှတစ်ဆင့် နောက် snapshot တွင် မည်သည့် ပြုပြင်ပြောင်းလဲမှုများ ပါဝင်သင့်သည်ကို သတ်မှတ်ခွင့်ပြုခြင်းဖြင့် ထိုကဲ့သို့သော အခြေအနေများကို ဖြည့်ဆည်းပေးသည်။

Git command-line interface

အချက်အလက်များ ထပ်နေခြင်းကို ရှောင်ရှားရန်အတွက် ဤသင်ခန်းစာ မှတ်စုများတွင် အောက်ပါ command များကို အသေးစိတ် ရှင်းပြမည် မဟုတ်ပါ။ ပိုမိုသိရှိလိုပါက လွန်စွာ အကြံပြုထားသော **Pro Git** (https://git-scm.com/book/en/v2) စာအုပ်ကို ဖတ်ရှုပါ သို့မဟုတ် သင်ခန်းစာ ဗီဒီယိုကို ကြည့်ရှုပါ။

အခြေခံများ

- `git help <command>` : git command တစ်ခုအတွက် အကူအညီ ရယူရန်
- `git init` : `.git` directory တွင် data များ သိမ်းဆည်းထားသည့် git repo အသစ်တစ်ခု ဖန်တီးရန်
- `git status` : မည်သည့်အရာများ ဖြစ်ပျက်နေသည်ကို ပြောပြရန်
- `git add <filename>` : ဖိုင်များကို staging area သို့ ထည့်သွင်းရန်
- `git commit` : commit အသစ်တစ်ခု ဖန်တီးရန်

```
- <a href="https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html" class="ext-link">ကောင်းမွန်သော commit မက်ဆေ့ဂျ်များ</a> <span class="ext-url-print">(https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html)</span> ရေးသားပါ!
- <a href="https://chris.beams.io/posts/git-commit/" class="ext-link">ကောင်းမွန်သော commit မက်ဆေ့ဂျ်များ</a> <span class="ext-url-print">(https://chris.beams.io/posts/git-commit/)</span> ရေးသားရန် နောက်ထပ် အကြောင်းရင်းများ!
```

- `git log` : မှတ်တမ်း၏ ပြန့်ပြူးသော log ကို ပြသရန်
- `git log --all --graph --decorate` : မှတ်တမ်းကို DAG အဖြစ် အမြင်ပုံဖော် ပြသရန်
- `git diff <filename>` : staging area နှင့် နှိုင်းယှဉ်၍ သင်ပြုလုပ်ခဲ့သော ပြောင်းလဲမှုများကို ပြသရန်
- `git diff <revision> <filename>` : snapshot များအကြား ဖိုင်တစ်ခု၏ ကွဲပြားမှုများကို ပြသရန်
- `git checkout <revision>` : HEAD ကို အပ်ဒိတ်လုပ်ရန် (branch တစ်ခုကို checkout လုပ်ပါက လက်ရှိ branch ကိုပါ အပ်ဒိတ်လုပ်သည်)

Branch ခွဲခြင်းနှင့် ပေါင်းစည်းခြင်း (Branching and merging)

- `git branch` : branch များကို ပြသရန်
- `git branch <name>` : branch တစ်ခု ဖန်တီးရန်
- `git switch <name>` : branch တစ်ခုသို့ ပြောင်းရန်
- `git checkout -b <name>` : branch တစ်ခု ဖန်တီးပြီး ထို branch သို့ ပြောင်းရန်

```
- `git branch <name>; git switch <name>` နှင့် အတူတူပင် ဖြစ်သည်
```

- `git merge <revision>` : လက်ရှိ branch ထဲသို့ ပေါင်းစည်းရန်
- `git mergetool` : merge conflict များကို ဖြေရှင်းရာတွင် ကူညီရန် အဆင့်မြင့် tool တစ်ခုကို အသုံးပြုရန်
- `git rebase` : patch အစုအဝေးကို base အသစ်တစ်ခုပေါ်သို့ rebase လုပ်ရန်

Remotes

- `git remote` : remote များကို စာရင်းပြုပြုရန်
- `git remote add <name> <url>` : add a remote
- `git push <remote> <local branch>:<remote branch>` : remote သို့ object များကို ပို့ရန်နှင့် remote reference ကို အပ်ဒိတ်လုပ်ရန်
- `git branch --set-upstream-to=<remote>/<remote branch>` : local branch နှင့် remote branch အကြား ချိတ်ဆက်မှုကို သတ်မှတ်ရန်
- `git fetch` : remote မှ objects/references များကို ရယူရန်
- `git pull` : `git fetch; git merge` နှင့် အတူတူပင် ဖြစ်သည်
- `git clone` : remote မှ repository ကို ဒေါင်းလုဒ်လုပ်ရန်

ပြန်လည်ရုပ်သိမ်းခြင်း (Undo)

- `git commit --amend` : commit တစ်ခု၏ ပါဝင်သည့်အရာများ/မက်ဆေ့ဂျ်ကို ပြင်ဆင်ရန်
- `git reset <file>` : ဖိုင်တစ်ခုကို unstage ပြုလုပ်ရန်
- `git restore` : ပြောင်းလဲမှုများကို ပယ်ဖျက်ရန်

အဆင့်မြင့် Git

- `git config` : Git သည် စိတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်း အလွန်မြင့်မားသည် (https://git-scm.com/docs/git-config)
- `git clone --depth=1` : version history အပြည့်အစုံ မပါဘဲ shallow clone လုပ်ရန်
- `git add -p` : တုံ့ပြန်မှုပါသော (interactive) staging ပြုလုပ်ရန်
- `git rebase -i` : တုံ့ပြန်မှုပါသော (interactive) rebasing ပြုလုပ်ရန်
- `git blame` : မည်သည့်လိုင်းကို မည်သူ နောက်ဆုံး ပြင်ဆင်ခဲ့သည်ကို ပြသရန်
- `git stash` : working directory ၏ ပြောင်းလဲမှုများကို ခဏတာ ဖယ်ရှားထားရန်
- `git bisect` : မှတ်တမ်းကို binary search ဖြင့် ရှာဖွေရန် (ဥပမာ - တိုးတက်မှု နောက်ပြန်ဆုတ်သွားသည့် အမှားများ (regressions) ကို ရှာရန်)
- `git revert` : ယခင် commit ၏ အကျိုးသက်ရောက်မှုကို နောက်ကြောင်းပြန်လှည့်ပေးသည့် commit အသစ်တစ်ခု ဖန်တီးရန်
- `git worktree` : တစ်ချိန်တည်းတွင် branch အများအပြားကို checkout လုပ်ရန်
- `.gitignore` : စောင့်ကြည့် မှတ်တမ်းမတင်ဘဲ လျစ်လျူရှုမည့် ဖိုင်များကို သတ်မှတ်ရန် (https://git-scm.com/docs/gitignore)

အထွေထွေ

- **GUIs**: Git အတွက် GUI clients (https://git-scm.com/downloads/guis) အများအပြား ရှိပါသည်။ ကျွန်ုပ်တို့ ကိုယ်တိုင်ကမူ ၎င်းတို့ကို မသုံးဘဲ command-line interface ကိုသာ အသုံးပြုပါသည်။
- **Shell integration**: သင့် shell prompt ၏ အစိတ်အပိုင်းတစ်ခုအဖြစ် Git status ပါဝင်နေခြင်းသည် အလွန် အဆင်ပြေပါသည် (zsh (https://github.com/olivierverdier/zsh-git-prompt)၊ bash (https://github.com/magicmonty/bash-git-prompt))။ Oh My Zsh (https://github.com/ohmyzsh/ohmyzsh) ကဲ့သို့သော framework များတွင် မကြာခဏ ပါဝင်လေ့ရှိသည်။
- **Editor integration**: အထက်ပါအတိုင်းပင် စွမ်းဆောင်ရည်များစွာ ပါဝင်သော အဆင်ပြေသည့် စုစည်းချိတ်ဆက်မှုများ ဖြစ်သည်။ fugitive.vim (https://github.com/tpope/vim-fugitive) သည် Vim အတွက် စံနှုန်းတစ်ခု ဖြစ်သည်။
- **Workflows**: ကျွန်ုပ်တို့သည် သင်အား data model နှင့်အတူ အခြေခံ command အချို့ကို သင်ကြားပေးခဲ့ပြီး ဖြစ်သည်၊ ပရောဂျက်ကြီးများတွင် အလုပ်လုပ်သည့်အခါ မည်သည့် လေ့ကျင့်ဆောင်ရွက်မှုများကို လိုက်နာရမည်ဆိုသည်ကိုမူ မပြောပြရသေးပါ (ကွဲပြားသော (https://nvie.com/posts/a-successful-git-branching-model/) ချဉ်းကပ်နည်းများ (https://www.endoflineblog.com/gitflow-considered-harmful) များစွာ (https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) ရှိပါသည်)။
- **GitHub**: Git သည် GitHub မဟုတ်ပါ။ GitHub တွင် အခြားပရောဂျက်များသို့ ကုဒ် ပံ့ပိုးပေးနိုင်သည့် pull requests (https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests) ဟုခေါ်သော သီးခြား နည်းလမ်းတစ်ခု ရှိသည်။
- **အခြား Git ဝန်ဆောင်မှုပေးသူများ**: GitHub သည် သီးသန့် အထူးဖြစ်မနေပါ - GitLab (https://about.gitlab.com/) နှင့် BitBucket (https://bitbucket.org/) ကဲ့သို့သော Git repository လက်ခံသိမ်းဆည်းပေးသည့် ဝန်ဆောင်မှုများစွာ ရှိပါသည်။

လေ့လာရန် အရင်းအမြစ်များ

- **Pro Git** (https://git-scm.com/book/en/v2) သည် **အထူး အကြံပြုထားသော ဖတ်ရှုရန် စာအုပ်** ဖြစ်သည်။ ယခု အခါ သင်သည် data model ကို နားလည်သွားပြီဖြစ်သောကြောင့် အခန်း ၁ မှ ၅ ထိ ဖတ်ရှုလိုက်ပါက Git ကို ကျွမ်းကျင်စွာ အသုံးပြုရန် လိုအပ်သမျှ၏ အစိတ်အပိုင်းအများစုကို သင်ကြားပေးမည် ဖြစ်သည်။ နောက်ပိုင်း အခန်းများတွင် စိတ်ဝင်စားဖွယ်ရာ အဆင့်မြင့် အကြောင်းအရာများ ပါဝင်သည်။
- **Oh Shit, Git!?!** (https://ohshitgit.com/) သည် အတွေ့များသော Git အမှားများမှ မည်သို့ ပြန်လည်ပြင်ဆင်ရမည်ကို ဖော်ပြထားသည့် လမ်းညွှန်တိုတစ်ခု ဖြစ်သည်။
- **Git for Computer Scientists** (https://eagain.net/articles/git-for-computer-scientists/) သည် ဤသင်ခန်းစာမှတ်စုများထက် pseudocode လျှော့၍ အဆင့်မြင့် ပုံကြမ်းများ ပိုမိုပါဝင်သော Git ၏ data model အကြောင်း ရိုးရှင်းသော ရှင်းလင်းချက် ဖြစ်သည်။
- **Git from the Bottom Up** (https://jwiegley.github.io/git-from-the-bottom-up/) သည် ပိုမို စိတ်ဝင်စားသူများအတွက် data model အပြင် Git ၏ အကောင်အထည်ဖော်မှု အသေးစိတ်များကို အသေးစိတ် ရှင်းပြထားခြင်း ဖြစ်သည်။
- **How to explain git in simple words** (https://smusamashah.github.io/blog/2017/10/14/explain-git-in-simple-words)
- **Learn Git Branching** (https://learngitbranching.js.org/) သည် သင့်အား Git အကြောင်း သင်ကြားပေးသည့် browser အခြေပြု ဂိမ်းတစ်ခု ဖြစ်သည်။

လေ့ကျင့်ခန်းများ

1. သင့်တွင် ယခင်က Git အတွေ့အကြုံ မရှိပါက **Pro Git** (https://git-scm.com/book/en/v2) ၏ ပထမ အခန်း အနည်းငယ်ကို ဖတ်ကြည့်ပါ သို့မဟုတ် **Learn Git Branching** (https://learngitbranching.js.org/) ကဲ့သို့သော သင်ခန်းစာကို လေ့လာပါ။ လေ့ကျင့် လုပ်ဆောင်နေချိန်တွင် Git command များကို data model နှင့် ဆက်စပ်ကြည့်ပါ။
2. **သင်တန်းဝင်ဆိုက်အတွက် repository** (https://github.com/missing-semester/missing-semester) ကို clone လုပ်ပါ။

```
1. Version history ကို graph အဖြစ် အမြင်ပုံဖော်ကြည့်ခြင်းဖြင့် လေ့လာစုံစမ်းပါ။
1. `README.md` ကို နောက်ဆုံး ပြင်ဆင်ခဲ့သူမှာ မည်သူနည်း။ (လမ်းညွှန် - argument ပါဝင်သော `git log` ကို အသုံးပြုပါ)။
1. `_config.yml` ၏ `collections:` လိုင်းသို့ နောက်ဆုံး ပြင်ဆင်မှုနှင့် သက်ဆိုင်သည့် commit မက်ဆေ့ဂျ်မှာ မည်သည့်အရာ ဖြစ်သနည်း။ (လမ်းညွှန် - `git blame` နှင့် `git show` ကို အသုံးပြုပါ)။
```

1. Git ကို လေ့လာရာတွင် တွေ့ရလေ့ရှိသော အမှားတစ်ခုမှာ Git ဖြင့် မထိန်းချုပ်သင့်သော ဖိုင်ကြီးများကို commit လုပ်မိခြင်း သို့မဟုတ် လျှို့ဝှက်ချက် အချက်အလက်များကို ထည့်သွင်းမိခြင်း ဖြစ်သည်။ ဖိုင်တစ်ခုကို repository သို့ ထည့်သွင်းကြည့်ပါ၊ commit အချို့ ပြုလုပ်ပြီး ထိုဖိုင်ကို *history* မှ (နောက်ဆုံး commit တစ်ခုတည်းမှ မဟုတ်ဘဲ) ဖျက်ပစ်ပါ။ သင်သည် **ဤနေရာ** (https://help.github.com/articles/removing-sensitive-data-from-a-repository/) ကို ကြည့်ရှုလိုပေမည်။
2. GitHub မှ repository တစ်ခုကို clone လုပ်ပြီး ၎င်း၏ လက်ရှိ ဖိုင်တစ်ခုကို ပြင်ဆင်ပါ။ `git stash` လုပ်သည့်အခါ မည်သို့ ဖြစ်ပျက်သနည်း။ `git log --all --oneline` ကို runသည့်အခါ မည်သည့် အရာကို တွေ့ရသနည်း။ `git stash` ဖြင့် သင်ပြုလုပ်ခဲ့သည်ကို ပြန်ဖြုတ်ရန် `git stash pop` ကို runပါ။ မည်သည့် အခြေအနေမျိုးတွင် ဤသည် အသုံးဝင်နိုင်သနည်း။
3. Command line tools အများအပြားကဲ့သို့ပင် Git သည် `~/.gitconfig` ဟုခေါ်သော configuration file (သို့မဟုတ် dotfile) ကို ပံ့ပိုးပေးထားသည်။ `~/.gitconfig` တွင် alias တစ်ခု ဖန်တီးပါ၊ သို့မှသာ သင် သည် `git graph` ကို runသည့်အခါ `git log --all --graph --decorate --oneline` ၏ output ကို ရရှိမည်ဖြစ်သည်။ ဤအရာကို `~/.gitconfig` ဖိုင်ကို တိုက်ရိုက် **ပြင်ဆင်ခြင်း** (https://git-scm.com/docs/git-config#Documentation/git-config.txt-alias) ဖြင့် ပြုလုပ်နိုင်သည် သို့မဟုတ် alias ထည့်သွင်းရန် `git config` command ကို အသုံးပြုနိုင်သည်။ Git alias များအကြောင်း အချက်အလက်များကို **ဤ နေရာ** (https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases) တွင် တွေ့နိုင်ပါသည်။

4. `git config --global core.excludesfile ~/.gitignore_global` ကို runပြီးနောက် `~/.gitignore_global` တွင် အထွေထွေ (global) ignore pattern များကို သတ်မှတ်နိုင်သည်။ ဤသည်မှာ Git အသုံးပြုမည့် global ignore file ၏ တည်နေရာကို သတ်မှတ်ပေးခြင်း ဖြစ်သော်လည်း ထိုလမ်းကြောင်းတွင် ဖိုင်ကို ကိုယ်တိုင် ဖန်တီးရန် လိုအပ်နေသေးသည်။ OS သို့မဟုတ် editor သီးသန့် ယာယီဖိုင်များဖြစ်သော `.DS_Store` ကဲ့သို့သော အရာများကို လျစ်လျူရှုရန် သင့် global gitignore file ကို သတ်မှတ်ပါ။

5. [သင်တန်းဝင်ဆိုက်အတွက် repository](https://github.com/missing-semester/missing-semester) (https://github.com/missing-semester/missing-semester) ကို Fork လုပ်ပါ၊ စာလုံးပေါင်းမှားခြင်း သို့မဟုတ် အခြား တိုးတက်ကောင်းမွန်အောင် ပြုလုပ်နိုင်သည့်အရာတစ်ခုကို ရှာဖွေပြီး GitHub တွင် pull request တစ်ခု တင်သွင်းပါ (ဤနေရာ (https://github.com/firstcontributions/first-contributions) ကို ကြည့်ရှုလိုပေမည်)။ ကျေးဇူးပြု၍ အသုံးဝင်သော PR များကိုသာ တင်သွင်းပါ (ကျွန်ုပ်တို့ထံ spam မလုပ်ပါနှင့်!)။ ပြုလုပ်ရန် တိုးတက်ကောင်းမွန်မှု မရှာတွေ့ပါက ဤလေ့ကျင့်ခန်းကို ကျော်သွားနိုင်ပါသည်။

6. ပူးပေါင်း လုပ်ဆောင်သည့် အခြေအနေတစ်ခုကို အတုယူ ဖန်တီးခြင်းဖြင့် merge conflict များကို ဖြေရှင်းခြင်းကို လေ့ကျင့်ပါ -

```
1. `git init` ဖြင့် repository အသစ်တစ်ခု ဖန်တီးပြီး စာကြောင်းအနည်းငယ် ပါဝင်သော `recipe.
txt` ဟုခေါ်သော ဖိုင်တစ်ခု ဖန်တီးပါ (ဥပမာ - ရိုးရှင်းသော ဟင်းချက်နည်းတစ်ခု)။
1. ၎င်းကို commit လုပ်ပါ၊ ထို့နောက် branch နှစ်ခု ဖန်တီးပါ - `git branch salty` နှင့် `git bra
nch sweet`။
1. `salty` branch တွင် စာကြောင်းတစ်လိုင်းကို ပြင်ဆင်ပါ (ဥပမာ - "1 cup sugar" မှ "1 cup s
alt" သို့ ပြောင်းပါ) ပြီးလျှင် commit လုပ်ပါ။
1. `sweet` branch တွင် ထိုစာကြောင်းကိုပင် မတူညီဘဲ ပြင်ဆင်ပါ (ဥပမာ - "1 cup sugar" မှ "2
cups sugar" သို့ ပြောင်းပါ) ပြီးလျှင် commit လုပ်ပါ။
1. ယခု `master` သို့ ပြောင်းပြီး `git merge salty`၊ ထို့နောက် `git merge sweet` ကို စမ်းကြ
ည့်ပါ။ မည်သို့ ဖြစ်ပျက်သနည်း။ `recipe.txt` ၏ ပါဝင်သည့်အရာများကို ကြည့်ပါ - `<<<<<<<`၊ `==
=====` နှင့် `>>>>>>>` အမှတ်အသားများသည် မည်သည့်အရာကို ဆိုလိုသနည်း။
1. သင်လိုချင်သော အကြောင်းအရာကို သိမ်းဆည်းရန် ဖိုင်ကို ပြင်ဆင်ခြင်း၊ conflict အမှတ်အသားများကို
ဖျက်ထုတ်ခြင်းနှင့် `git add` နှင့် `git commit` (သို့မဟုတ် `git merge --continue`) တို့ဖြင့် ပ
ေါင်းစည်းခြင်းကို ပြီးမြောက်စေခြင်းဖြင့် conflict ကို ဖြေရှင်းပါ။ သို့မဟုတ်ဘဲ graphical သို့မဟုတ် t
erminal အခြေပြု merge tool တစ်ခုဖြင့် conflict ကို ဖြေရှင်းရန် `git mergetool` ကို အသုံးပြု
ကြည့်ပါ။
1. သင် ယခုလေးတင် ဖန်တီးခဲ့သော merge history ကို အမြင်ပုံဖော်ရန် `git log --graph --onel
ine` ကို အသုံးပြုပါ။
```

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. XKCD comic	https://xkcd.com/1597/	
2. xkcd 1597	https://imgs.xkcd.com/comics/git.png	
3. SHA-1 hash	https://en.wikipedia.org/wiki/SHA-1	
4. Pro Git	https://git-scm.com/book/en/v2	
5. ကောင်းမွန်သော commit မက်ဆေ့ဂျ်များ	https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html	
6. ကောင်းမွန်သော commit မက်ဆေ့ဂျ်များ	https://chris.beams.io/posts/git-commit/	
7. စီတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်းအလွန်မြင့်မားသည်	https://git-scm.com/docs/git-config	
8. သတ်မှတ်ရန်	https://git-scm.com/docs/gitignore	
9. GUI clients	https://git-scm.com/downloads/guis	
10. zsh	https://github.com/olivierverdier/zsh-git-prompt	
11. bash	https://github.com/magicmonty/bash-git-prompt	
12. Oh My Zsh	https://github.com/ohmyzsh/ohmyzsh	
13. fugitive.vim	https://github.com/tpope/vim-fugitive	
14. ကွဲပြားသော	https://nvie.com/posts/a-successful-git-branching-model/	
15. ချဉ်းကပ်နည်းများ	https://www.endoflineblog.com/gitflow-considered-harmful	

Resource / Description	URL	Micro QR
16. များစွာ	https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow	
17. pull requests	https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests	
18. GitLab	https://about.gitlab.com/	
19. BitBucket	https://bitbucket.org/	
20. Oh Shit, Git!?!	https://ohshitgit.com/	
21. Git for Computer Scientists	https://eagain.net/articles/git-for-computer-scientists/	
22. Git from the Bottom Up	https://jwiegley.github.io/git-from-the-bottom-up/	
23. How to explain git in simple words	https://smusamashah.github.io/blog/2017/10/14/explain-git-in-simple-words	
24. Learn Git Branching	https://learngitbranching.js.org/	
25. သင်တန်းဝင်ဆိုက်အတွက် repository	https://github.com/missing-semester/missing-semester	
26. ဤနေရာ	https://help.github.com/articles/removing-sensitive-data-from-a-repository/	
27. ပြင်ဆင်ခြင်း	https://git-scm.com/docs/git-config#Documentation/git-config.txt-alias	
28. ဤနေရာ	https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases	
29. ဤနေရာ	https://github.com/firstcontributions/first-contributions	

လိုင်စင်နှင့် မူပိုင်ခွင့် (License)

မူပိုင်ခွင့် လိုင်စင် (License)

ဤသင်တန်းရှိ ဝဘ်ဆိုက် source code၊ သင်ခန်းစာ မှတ်တမ်းများ၊ လေ့ကျင့်ခန်းများနှင့် သင်ခန်းစာ ဗီဒီယို များ အပါအဝင် အကြောင်းအရာ အားလုံးကို CC BY-NC-SA 4.0 လိုင်စင်ဖြင့် ထုတ်ဝေထားပါသည်။

ဤသည်မှာ သင်၏ အခွင့်အရေးများ ဖြစ်ကြပါသည် - - **Share** — မည်သည့် မီဒီယာ သို့မဟုတ် မူဘောင် ဖြင့် မဆို အကြောင်းအရာများကို ကူးယူ ပြန်လည် ဖြန့်ဝေနိုင်ပါသည်။ - **Adapt** — အကြောင်းအရာများကို ပြုပြင် ပြောင်းလဲခြင်း၊ တည်ဆောက်ခြင်းများ ပြုလုပ်နိုင်ပါသည်။

အောက်ပါ စည်းကမ်းချက်များနှင့် အညီ ပြုလုပ်ရမည် -

- **Attribution** — သင်သည် မူရင်း ဖန်တီးသူအား သင့်တော်သော အသိအမှတ်ပြုမှု ပေးရမည်၊ လိုင်စင်သို့ လင့်ခ် ချိတ်ဆက်ပေးရမည်၊ ပြောင်းလဲမှုများ ပြုလုပ်ထားပါက ညွှန်းဆိုပေးရမည်။
- **NonCommercial** — ဤ အကြောင်းအရာများကို စီးပွားရေး ရည်ရွယ်ချက်ဖြင့် အသုံးပြုပိုင်ခွင့် မရှိပါ။
- **ShareAlike** — အကယ်၍ သင်သည် ဤ အကြောင်းအရာများကို ပြုပြင် ပြောင်းလဲပါက၊ သင်၏ ပံ့ပိုးမှု များကို မူရင်း လိုင်စင်အတိုင်း အတိအကျ ပြန်လည် ဖြန့်ဝေရမည်။

ဤသည်မှာ မူပိုင်ခွင့် Legal Code ၏ လူနားလည်လွယ်သော အတိုချုပ် ဖြစ်ပါသည်။

Contribution guidelines

ကျွန်ုပ်တို့၏ GitHub repo တွင် Issues နှင့် Pull Requests တင်သွင်းခြင်းဖြင့် သင်ခန်းစာ အကြောင်းအရာ များအတွက် ပြင်ဆင်ချက်များနှင့် အကြံပြုချက်များကို တင်သွင်းနိုင်ပါသည်။ ၎င်းတွင် ဗီဒီယို စာတန်းထိုး များလည်း ပါဝင်ပါသည် (ဒီမှာ ကြည့်ပါ)။

Translation guidelines

မူပိုင်ခွင့် စည်းကမ်းချက်များကို လိုက်နာသရွေ့ သင်ခန်းစာ မှတ်တမ်းများနှင့် လေ့ကျင့်ခန်းများကို လွတ်လပ် စွာ ဘာသာပြန်ဆိုနိုင်ပါသည်။ သင်၏ ဘာသာပြန်ဆိုမှုသည် သင်တန်း ပုံစံအတိုင်း ဖြစ်ပါက၊ ကျွန်ုပ်တို့၏ စာမျက်နှာမှတဆင့် သင်၏ ဘာသာပြန် မူကွဲကို လင့်ခ် ချိတ်ဆက်ပေးနိုင်ရန် ကျွန်ုပ်တို့ထံ ဆက်သွယ်ပါ။

ဗီဒီယို စာတန်းထိုးများ ဘာသာပြန်ဆိုရန်အတွက် YouTube တွင် Community contribution အဖြစ် တင် သွင်းပေးပါ။

MIT CSIL • BURMESE EBOOK EDITION

The Missing Semester of Your CS Education

(2026 Burmese Edition)

ORIGINAL COURSE

Originally designed and taught at Massachusetts Institute of Technology (MIT) by **Anish Athalye, Jon Gjengset, and Jose Javier Gonzalez Ortiz**.

Original Website: https://missing.csail.mit.edu/

MY LOCALIZATION & PUBLICATION

Burmese translation and digital eBook publication maintained by **FOSS Myanmar** (fossmyanmar.org) and **Vibe Code Tours** (vibecode.tours).

Burmese Website: https://missing-semester-my.github.io/

Scan to Visit Burmese Course Portal:

https://missing-semester-my.github.io/