MIT CSIL • COMPUTER SCIENCE EDUCATION

2020 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)

- **Command-line Environment**
- **Course Overview + The Shell**
- **ဒေတာ ပြုပြင်စီမံခြင်း (Data Wrangling)**
- **Debugging and Profiling**
- **Editors (Vim)**
- **2020 Lectures**
- **Metaprogramming**
- **အထွေထွေ ခေါင်းစဉ်များ (Potpourri)**
- **Q&A**
- **Security and Cryptography**
- **Shell Tools နှင့် Scripting**
- **Version Control (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,

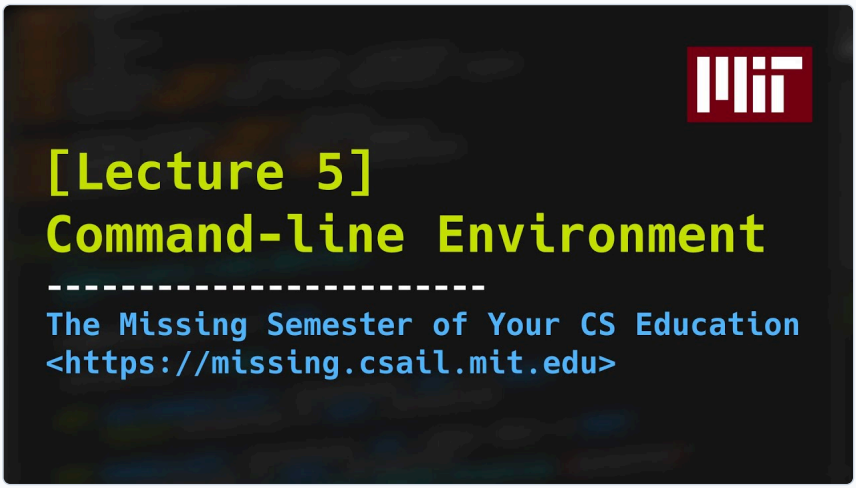
Anish၊ **Jon** နှင့် **Jose**

Command-line Environment

အနှစ်ချုပ် - Job controlı terminal multiplexersı dotfiles နှင့် SSH အသုံးပြု၍ remote machines များ လုပ်ဆောင်ပုံ အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Command-line Environment



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=e8BO_dYxk5c

ဤသင်ခန်းစာတွင် Shell ကို အသုံးပြုရာ၌ သင်၏ လုပ်ငန်းစဉ် (workflow) ကို ပိုမိုကောင်းမွန် လျင်မြန်စေ မည့် နည်းလမ်းများစွာကို အသေးစိတ် လေ့လာသွားပါမည်။ ကျွန်ုပ်တို့သည် Shell ကို အသုံးပြုလာခဲ့သည်မှာ အတော်အတန် ကြာမြင့်ပြီ ဖြစ်သော်လည်း အဓိကအားဖြင့် Command များကို တစ်ခုပြီးတစ်ခု စတင် လုပ်ဆောင် (execute) ခြင်းအပေါ်၌သာ အာရုံစိုက်ခဲ့ကြပါသည်။ ယခုအခါတွင် Process အများအပြားကို တစ်ပြိုင်နက်တည်း မည်သို့ စောင့်ကြည့် စီမံမလဲ၊ သီးခြား Process တစ်ခုခုကို မည်သို့ ခေတ္တရပ်ဆိုင်း သို့မဟုတ် ပိတ်ပစ်မလဲ နှင့် Process များကို နောက်ကွယ် (background) တွင် မည်သို့ run (run) ထားမလဲ ဆိုသည်များကို လေ့လာသွားမည် ဖြစ်ပါသည်။

ထို့ပြင် Aliases များကို သတ်မှတ်ခြင်းနှင့် Dotfiles များကို အသုံးပြု၍ ပြင်ဆင်သတ်မှတ်ခြင်း (configuration) ဖြင့် သင်၏ Shell နှင့် အခြား Tool များကို မည်သို့ ပိုမိုကောင်းမွန်အောင် ပြုလုပ်နိုင်သည်ကို လည်း လေ့လာပါမည်။ ဤနည်းလမ်းနှစ်ခုစလုံးသည် သင်၏ စက်အားလုံးတွင် တူညီသော Configuration များကို အသုံးပြုနိုင်စေပြီး ရှည်လျားသော Command များကို ထပ်ခါတလဲလဲ ရိုက်နှိပ်စရာ မလိုဘဲ အချိန်ကုန် သက်သာစေပါသည်။ ထို့အပြင် SSH ကို အသုံးပြု၍ အဝေးရှိ စက်များ (remote machines) နှင့် မည်သို့ ချိတ်ဆက် အလုပ်လုပ်ရမည်ကိုပါ လေ့လာသွားမည် ဖြစ်ပါသည်။

Job Control

အချို့သော အခြေအနေများတွင် Command တစ်ခု runနေစဉ်အတွင်း ယင်း၏ လုပ်ဆောင်ချက်ကို ခေတ္တရပ်ရန် သို့မဟုတ် ဖြတ်တောက်ရန် လိုအပ်တတ်ပါသည်။ အထူးသဖြင့် ဖိုင်လမ်းကြောင်း အမြောက်အများကို ရှာဖွေနေသည့် `find` ကဲ့သို့သော Command များသည် အချိန်အလွန် ကြာမြင့်နေသည့် အခါမျိုး ဖြစ်ပါသည်။ များသောအားဖြင့် `Ctrl-C` ကို နှိပ်လိုက်ပါက Command ၏ လုပ်ဆောင်ချက် ရပ်တန့်သွားမည် ဖြစ်သည်။ သို့သော် ဤလုပ်ဆောင်ချက်သည် နောက်ကွယ်တွင် မည်သို့ အလုပ်လုပ်သနည်း၊ အဘယ်ကြောင့် အချို့သော Process များသည် ရပ်တန့်မသွားဘဲ ဖြစ်နေရသနည်း။

Killing a process

သင်၏ Shell သည် Process များထံ အချက်အလက်များ ပေးပို့ဆက်သွယ်ရန်အတွက် UNIX Communication Mechanism တစ်ခုဖြစ်သည့် *Signal* ကို အသုံးပြုပါသည်။ Process တစ်ခုသည် Signal တစ်ခုကို လက်ခံရရှိသောအခါ ယင်း၏ လုပ်ဆောင်နေမှုကို ခေတ္တရပ်ဆိုင်းပြီး လက်ခံရရှိသည့် Signal ၏ ညွှန်ကြားချက်အတိုင်း ဆက်လက် လုပ်ဆောင်ပါသည်။ ထို့ကြောင့် Signal များကို *Software Interrupts* ဟု ခေါ်ဆိုကြပါသည်။

ကွန်ပျူတာ၏ ကီးစွတွင် `Ctrl-C` ကို နှိပ်လိုက်သောအခါ Shell သည် Process ထံသို့ `SIGINT` ဆိုသည့် Signal ကို ပေးပို့လိုက်ခြင်း ဖြစ်သည်။

အောက်တွင် `SIGINT` Signal ကို ဖမ်းယူပြီး ပစ်ပယ်ထားကာ (ignore) ဆက်လက် runနေမည့် မူလ Python ပရိုဂရမ် အသေးစားလေးကို ဖော်ပြထားပါသည်။ ဤပရိုဂရမ်ကို ပိတ်ပစ်ရန်အတွက် `Ctrl-C` အစား `Ctrl-\` ကို နှိပ်၍ `SIGQUIT` Signal ကို အသုံးပြုရမည် ဖြစ်သည်။

```
#!/usr/bin/env python
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
```

ဤပရိုဂရမ်သို့ `SIGINT` ကို နှစ်ကြိမ်တိုင်တိုင် ပေးပို့ပြီးနောက် `SIGQUIT` ပေးပို့လိုက်သည့်အခါ မည်သို့ဖြစ်ပျက်သွားသည်ကို အောက်တွင် ကြည့်နိုင်ပါသည်။ Terminal တွင် `^` သင်္ကေတသည် `Ctrl` ခလုတ်ကို နှိပ်ထားခြင်းကို ကိုယ်စားပြုပါသည်။

```
$ 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` နှစ်ခုစလုံးသည် Terminal နှင့် ပတ်သက်သော တောင်းဆိုချက်များအတွက် အသုံးပြုလေ့ရှိသော်လည်း၊ Process တစ်ခုကို စနစ်တကျ ပိတ်ပစ်ရန် (exit gracefully) အတွက် ပိုမို ယေဘုယျကျသော 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>` ဟူသော Syntax ဖြင့် အသုံးပြုနိုင်ပါသည်။

Pausing and backgrounding processes

Signal များသည် Process တစ်ခုကို ပိတ်ပစ်ရုံသာမက အခြားသော လုပ်ဆောင်ချက်များကိုလည်း ပြုလုပ်နိုင်ပါသည်။ ဥပမာအားဖြင့် `SIGSTOP` သည် Process တစ်ခုကို ခေတ္တရပ်ဆိုင်း (pause) စေပါသည်။ Terminal တွင် `Ctrl-Z` ကို နှိပ်လိုက်ပါက Shell သည် Terminal Stop ကို ကိုယ်စားပြုသော `SIGTSTP` Signal ကို ပေးပို့မည် ဖြစ်သည်။

ထို့နောက် ခေတ္တရပ်ဆိုင်းထားသော Job ကို ရှေ့ကွယ် (foreground) တွင် ဆက်လက် runလိုပါက `fg` (https://www.man7.org/linux/man-pages/man1/fg.1p.html) ကိုလည်းကောင်း၊ နောက်ကွယ် (background) တွင် runလိုပါက `bg` (https://man7.org/linux/man-pages/man1/bg.1p.html) ကိုလည်းကောင်း အသီးသီး အသုံးပြုနိုင်ပါသည်။

`jobs` (https://www.man7.org/linux/man-pages/man1/jobs.1p.html) Command သည် လက်ရှိ Terminal Session တွင် မပြီးသေးဘဲ ရှိနေသော Job များ၏ စာရင်းကို ပြသပေးပါသည်။ ထို Job များကို ယင်းတို့၏ PID ဖြင့် ညွှန်းဆို နိုင်ပါသည် (PID ကို ရှာရန် `pgrep` (https://www.man7.org/linux/man-pages/man1/pgrep.1.html) ကို အသုံးပြုနိုင်သည်)။ ပို့မို့ လွယ်ကူသော နည်းလမ်းမှာ ရာခိုင်နှုန်းသင်္ကေတ `%` နောက်တွင် Job နံပါတ် (`jobs` က ပြသပေးသော နံပါတ်) ကို တွဲဖက်၍ ညွှန်းဆိုခြင်း ဖြစ်သည်။ နောက်ဆုံး Background လုပ်ထားသော Job ကို ညွှန်းဆိုလိုပါက `$!` ဟူသော Special Parameter ကို အသုံးပြုနိုင်ပါသည်။

အခြား သိရှိထားသင့်သည့် အချက်တစ်ခုမှာ Command ၏ နောက်တွင် `&` သင်္ကေတ ထည့်သွင်းလိုက်ပါက ထို Command သည် Background တွင် စတင် runမည်ဖြစ်ပြီး Prompt ထံသို့ ချက်ချင်း ပြန်ရောက်သွားမည် ဖြစ်သည်။ (သို့သော် output များသည် Terminal ၏ STDOUT သို့ ထွက်ပေါ်လာနိုင်သဖြင့် လိုအပ်ပါက Shell Redirections များကို အသုံးပြုပါ)။

စတင် runနေပြီးသား ပရိုဂရမ်တစ်ခုကို Background သို့ ပို့လိုပါက `Ctrl-Z` နှိပ်ပြီးနောက် `bg` ဟု ရိုက်နှိပ်နိုင်ပါသည်။ သတိပြုရန်မှာ Background ရောက်သွားသော Process များသည် သင်၏ Terminal ၏ Child Process များအဖြစ် ရှိနေဆဲ ဖြစ်သဖြင့် Terminal ကို ပိတ်လိုက်ပါက ထို Process များပါ ပိတ်သွားမည် ဖြစ်သည် (`SIGHUP` Signal ပေးပို့သွားမည် ဖြစ်သည်)။ ထိုသို့ မဖြစ်စေရန်အတွက် ပရိုဂရမ်ကို `nohup` (https://www.man7.org/linux/man-pages/man1/nohup.1.html) (`SIGHUP` ကို ပစ်ပယ်ရန် အသုံးပြုသည့် wrapper) ဖြင့် စတင် runနိုင်သည်၊ သို့မဟုတ် Process စတင်ပြီးသွားပါက `disown` Command ကို အသုံးပြုနိုင်ပါသည်။ နောက်တစ်နည်းမှာ နောက်လာမည့် အပိုင်းတွင် ဖော်ပြမည့် Terminal Multiplexer ကို အသုံးပြုခြင်း ဖြစ်ပါသည်။

အောက်တွင် ဤအယူအဆများကို လက်တွေ့ ပြသထားသည့် Sample Session တစ်ခု ဖြစ်ပါသည်။

```
$ 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

$ bg %1
[1]  - 18653 continued  sleep 1000

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

$ kill -STOP %1
[1]  + 18653 suspended (signal)  sleep 1000

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

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

$ jobs
[2]  + running    nohup sleep 2000

$ kill -SIGHUP %2

$ jobs
[2]  + running    nohup sleep 2000

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

$ jobs
```

အထူး Signal တစ်ခုမှာ `SIGKILL` ဖြစ်ပါသည်။ အကြောင်းမှာ ယင်း Signal ကို Process များက ဖမ်းယူသို့မဟုတ် ပစ်ပယ်ထား၍ မရဘဲ ချက်ချင်း ပိတ်ပစ်မည် ဖြစ်သည်။ သို့သော် မိဘမဲ့ မိခင်မဲ့ Child Process

များ ကျန်ရစ်ခဲ့ခြင်းကဲ့သို့သော ဆိုးကျိုးများ ရှိနိုင်ပါသည်။

ဤ Signal များ အကြောင်းကို [ဒီမှာ](https://en.wikipedia.org/wiki/Signal_(IPC)) ပိုမို လေ့လာနိုင်သည်၊ သို့မဟုတ် Terminal တွင် `man signal` (https://www.man7.org/linux/man-pages/man7/signal.7.html) သို့မဟုတ် `kill -l` ဟု ရိုက်နှိပ်၍ ကြည့်ရှုနိုင်ပါသည်။

Terminal Multiplexers

Command Line Interface ကို အသုံးပြုသည့်အခါ တစ်ချိန်တည်းတွင် အလုပ်တစ်ခုထက်မက ပြုုံတူ လုပ်ဆောင်ချင်သည့် အခြေအနေမျိုး မကြာခဏ ကြုံတွေ့ရတတ်ပါသည်။ ဥပမာအားဖြင့် သင်၏ Editor နှင့် Program ကို ဘေးချင်းယှဉ်၍ runလိုသည့် အခါမျိုး ဖြစ်သည်။ Terminal Window အသစ်များကို ဖွင့်ခြင်း ဖြင့်လည်း ဤသို့ ပြုလုပ်နိုင်သော်လည်း Terminal Multiplexer ကို အသုံးပြုခြင်းသည် ပိုမို စုံလင်ဆန်းသစ် သော ဖြေရှင်းနည်း ဖြစ်ပါသည်။

`tmux` (https://www.man7.org/linux/man-pages/man1/tmux.1.html) ကဲ့သို့သော Terminal Multiplexer များသည် Terminal Window များကို Panes များနှင့် Tabs များအဖြစ် ခွဲခြားပေးနိုင်သောကြောင့် Shell Session အမြောက်အမြားနှင့် ပြုုံတူ ချိတ်ဆက် ဆောင်ရွက်နိုင်စေပါသည်။ ထို့အပြင် Terminal Multiplexer များသည် လက်ရှိ Terminal Session ကို ခေတ္တ ဖြုတ်ထားခဲ့ပြီး (detach) နောက်ပိုင်းမှ ပြန်လည် ချိတ်ဆက် (reattach) စေနိုင်ပါသည်။ ဤသည်မှာ အဝေးရှိ Remote Machine များနှင့် အလုပ်လုပ်ရာတွင် `nohup` ကဲ့သို့သော လှည့်ကွက်များကို အသုံးပြုစရာ မလိုဘဲ သင်၏ လုပ်ငန်းစဉ်ကို ပိုမို ချောမွေ့ စနစ်ကျစေပါသည်။

ယနေ့ခေတ်တွင် ရေပန်းအစားဆုံး Terminal Multiplexer မှာ `tmux` (https://www.man7.org/linux/man-pages/man1/tmux.1.html) ဖြစ်ပါသည်။ `tmux` သည် စိတ်ကြိုက် ပြင်ဆင်ဖန်တီးနိုင်စွမ်း မြင့်မားပြီး ၎င်း၏ Keybindings များကို အသုံးပြု၍ Tabs များနှင့် Panes များကို ဖန်တီးကာ လျင်မြန်စွာ ကူးပြောင်း အသုံးပြု နိုင်ပါသည်။

`tmux` ကို အသုံးပြုရန် ၎င်း၏ Keybinding များကို သိရှိထားရန် လိုအပ်ပါသည်။ Keybinding အားလုံးသည် `<C-b> x` ဟူသော ပုံစံမျိုး ရှိပြီး ယင်း၏ အဓိပ္ပာယ်မှာ (၁) `Ctrl+b` ကို နှိပ်ပါ၊ (၂) `Ctrl+b` ကို ပြန်လွှတ်ပါ၊ ထို့နောက် (၃) `x` ကို နှိပ်ပါ ဟု ဖြစ်ပါသည်။ `tmux` တွင် အောက်ပါ Object အဆင့်ဆင့် ရှိပါသည် -

- **Sessions** - Session ဆိုသည်မှာ Window တစ်ခု သို့မဟုတ် တစ်ခုထက်မက ပါဝင်သော သီးခြား စာပွဲ ပြင် (workspace) ဖြစ်သည်။

```
+ `tmux` ဟု ရိုက်နှိပ်ပါက Session အသစ်တစ်ခု စတင်မည်။
+ `tmux new -s NAME` ဟု ရိုက်နှိပ်ပါက အမည်သတ်မှတ်၍ Session စတင်မည်။
+ `tmux ls` သည် လက်ရှိ Session များကို စာရင်းပြပေးမည်။
+ `tmux` အတွင်း၌ `&lt;C-b&gt; d` ကို နှိပ်ပါက လက်ရှိ Session မှ ခေတ္တ ထွက်မည် (detach)။
+ `tmux a` သည် နောက်ဆုံး Session သို့ ပြန်လည် ချိတ်ဆက်မည် (attach)။ သီးခြား Session ကို ညွှန်းလိုပါက `-t` flag ကို အသုံးပြုနိုင်သည်။
```

- **Windows** - Browser သို့မဟုတ် Editor များမှ Tabs များနှင့် တူညီပြီး၊ Session တစ်ခုတည်း၏ သီးခြားမြင်ကွင်း အပိုင်းအစများ ဖြစ်ကြသည်။

```
+ `<C-b> c` သည် Window အသစ်တစ်ခု ဖန်တီးမည်။ Window ကို ပိတ်လိုပါက `<C-d>` နှိပ်၍ Shel
l ကို ပိတ်လိုက်ရုံ ဖြစ်သည်။
+ `<C-b> N` သည် _N_ မြောက် Window သို့ သွားမည်။ (Window များကို နံပါတ်စဉ် တပ်ထားသည်)
+ `<C-b> p` သည် ယခင် Window သို့ သွားမည်။
+ `<C-b> n` သည် နောက် Window သို့ သွားမည်။
+ `<C-b> ,` သည် လက်ရှိ Window ၏ အမည်ကို ပြောင်းလဲမည်။
+ `<C-b> w` သည် လက်ရှိ Window များကို စာရင်းပြပေးမည်။
```

- **Panes** - Vim splits များနှင့် တူညီပြီး Screen တစ်ခုတည်းတွင် Shell အများအပြားကို ဘေးချင်းယှဉ်၍ ကြည့်ရှုနိုင်စေသည်။

```
+ `<C-b> "` သည် လက်ရှိ Pane ကို အလျားလိုက် (horizontal) ခွဲခြားမည်။
+ `<C-b> %` သည် လက်ရှိ Pane ကို ဒေါင်လိုက် (vertical) ခွဲခြားမည်။
+ `<C-b> <direction>` သည် သတ်မှတ်ထားသော _direction_ (မြားခလုတ်များ) အတိုင်း အခြား
Pane သို့ ရွှေ့ပြောင်းမည်။
+ `<C-b> z` သည် လက်ရှိ Pane ကို မျက်နှာပြင်ပြည့် (zoom) ချဲ့မည် သို့မဟုတ် မူလအတိုင်း ပြန်လုပ်မ
ည်။
+ `<C-b> <a href="https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmu
x/" class="ext-link">` သည် Scrollback mode သို့ ဝင်မည်။ ထို့နောက် `<space>` နှိပ်၍ hig
hlight လုပ်ပြီး `<enter>` နှိပ်၍ copy ကူးနိုင်သည်။
+ `<C-b> <space>` သည် Pane ၏ အခင်းအကျင်း ပုံစံများကို အစဉ်လိုက် ပြောင်းလဲပေးမည်။
```

ပိုမို လေ့လာလိုပါက၊ [`tmux` အကြောင်း အတိုချုပ် သင်ခန်းစာ (https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) နှင့် မူလ `screen` command အကြောင်းပါ အသေးစိတ် ရှင်းပြထားသည့် ဤသင်ခန်းစာ (https://linuxcommand.org/lc3_adv_termmux.php) တို့ကို ဖတ်ရှုနိုင်ပါသည်။ ထို့အပြင် UNIX စနစ်အများစုတွင် မူလပါဝင်လေ့ရှိသည့် **`screen`** (https://www.man7.org/linux/man-pages/man1/screen.1.html) အကြောင်းကိုလည်း လေ့လာထားသင့်ပါသည်။

Aliases

အလွန်ရှည်လျားသော Command များ သို့မဟုတ် Flag များစွာ ပါဝင်သော Command များကို မကြာခဏ ရိုက်နှိပ်ရခြင်းသည် စိတ်ပျက်ဖွယ် ကောင်းလှပါသည်။ ထို့ကြောင့် Shell အများစုသည် *Aliasing* (အမည်တို သတ်မှတ်ခြင်း) ကို ထောက်ပံ့ပေးထားကြပါသည်။ Shell Alias ဆိုသည်မှာ ရှည်လျားသော Command တစ်ခု အတွက် သင်၏ Shell က အလိုအလျောက် အစားထိုးပေးမည့် အမည်တို (shorthand) တစ်ခု ဖြစ်ပါသည်။ ဥပမာအားဖြင့် Bash တွင် Alias တစ်ခု၏ ပုံစံမှာ အောက်ပါအတိုင်း ဖြစ်ပါသည် -

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

သတိပြုရန်မှာ ညီမျှခြင်း `=` ဘေးတွင် Space မပါရပါ၊ အကြောင်းမှာ **`alias`** (https://www.man7.org/linux/man-pages/man1/alias.1p.html) သည် Argument တစ်ခုတည်းသာ လက်ခံသည့် Shell Command တစ်ခုဖြစ်သော ကြောင့် ဖြစ်ပါသည်။

Alias များတွင် အသုံးဝင်သော အင်္ဂါရုပ်များစွာ ရှိပါသည် -

```
# အသုံးများသော Flag များအတွက် အမည်တို သတ်မှတ်ခြင်း
alias ll="ls -lh"

# မကြာခဏ ရိုက်ရသော Command များအတွက် အချိန်ကုန် သက်သာစေခြင်း
alias gs="git status"
alias gc="git commit"
alias v="vim"

# စာလုံးမှား ရိုက်မိခြင်းကို ကာကွယ်ပေးခြင်း
alias sl=ls

# မူလ Command များကို ပိုမိုကောင်းမွန်သော အမူအကျင့်များဖြင့် အစားထိုးခြင်း
alias mv="mv -i"           # -i သည် ထပ်ရေးမည်လားဟု မေးမြန်းမည်
alias mkdir="mkdir -p"     # -p သည် လိုအပ်သော parent directory များကို ဖန်တီးပေးမည်
alias df="df -h"           # -h သည် လူနားလည်လွယ်သော ဖိုင်ဆိုဒ် ပုံစံဖြင့် ပြပေးမည်

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

# Alias ကို ကျော်လွန်၍ မူလ Command ကို runလိုပါက ရှေ့တွင် \ ထည့်ပါ
\ls
# သို့မဟုတ် unalias ဖြင့် Alias ကို လုံးဝ ပယ်ဖျက်ပါ
unalias la

# Alias ၏ အဓိပ္ပာယ်သတ်မှတ်ချက်ကို ကြည့်လိုပါက alias command ဖြင့် ခေါ်ပါ
alias ll
# ll='ls -lh' ဟု ထွက်ပေါ်လာမည်
```

သတိပြုရန်မှာ Alias များသည် မူလအားဖြင့် Session ပိတ်လိုက်ပါက ပျောက်ပျက်သွားမည် ဖြစ်သည်။ Alias များကို အမြဲတမ်း တည်တံ့စေရန်အတွက် `.bashrc` သို့မဟုတ် `.zshrc` ကဲ့သို့သော Shell Startup Files များတွင် ထည့်သွင်းထားရမည် ဖြစ်ပါသည်။

Dotfiles

ပရိုဂရမ် အမြောက်အများကို *Dotfiles* ဟု ခေါ်ဆိုကြသည့် Plain-text ဖိုင်များဖြင့် Configuration ပြုလုပ်ကြပါသည် (ဖိုင်အမည်များသည် `.` ဖြင့် စတင်လေ့ရှိသောကြောင့် ဖြစ်သည်၊ ဥပမာ `~/.vimrc` ၊ ထို့ကြောင့် ယင်းတို့သည် `ls` ရိုက်သည့်အခါ မူလအားဖြင့် ပုန်းကွယ်နေကြသည်)။

Shell များသည် ထိုကဲ့သို့သော ဖိုင်များဖြင့် Configuration ပြုလုပ်ရသည့် ပရိုဂရမ်များ၏ ဥပမာတစ်ခု ဖြစ်ပါသည်။ Shell စတင်ချိန်တွင် ယင်း၏ Configuration များကို ရယူရန် ဖိုင်များစွာကို ဖတ်ရှုပါသည်။ Shell အမျိုးအစားပေါ် မူတည်၍၊ Login မုဒ် သို့မဟုတ် Interactive မုဒ် အပေါ် မူတည်၍ ဤ လုပ်ငန်းစဉ်တစ်ခုလုံး သည် အတော်အတန် ရှုပ်ထွေးနိုင်ပါသည်။ ဤခေါင်းစဉ်နှင့် ပတ်သက်၍ အလွန်ကောင်းမွန်သော ဆောင်းပါး ကို [ဒီမှာ](https://web.archive.org/web/20260329133158/https://blog.flowblok.id.au/2013-02/shell-startup-scripts.html) ဖတ်ရှုနိုင်ပါသည်။

`bash` အတွက်ဆိုလျှင် သင်၏ `.bashrc` သို့မဟုတ် `.bash_profile` ကို ပြင်ဆင်ခြင်းသည် စနစ်အများစုတွင် အဆင်ပြေပါသည်။ ဤနေရာတွင် ခုနက ဖော်ပြခဲ့သော Alias များ သို့မဟုတ် သင်၏ `PATH` Environment Variable ကို ပြင်ဆင်ခြင်းကဲ့သို့သော Shell စတင်ချိန်တွင် runလိုသည့် Command များကို ထည့်သွင်းနိုင်ပါသည်။ အမှန်စင်စစ် ပရိုဂရမ်များစွာသည် ယင်းတို့၏ Binary ဖိုင်များကို ရှာဖွေတွေ့ရှိနိုင်စေရန် အတွက် `export PATH="$PATH:/path/to/program/bin"` ကဲ့သို့သော လိုင်းတစ်ခုကို သင်၏ Shell Configuration ဖိုင်တွင် ထည့်သွင်းခိုင်းလေ့ ရှိကြပါသည်။

Dotfile များဖြင့် ပြင်ဆင်သတ်မှတ်နိုင်သော အခြား Tool အချို့၏ ဥပမာများမှာ -

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

သင်၏ Dotfile များကို မည်သို့ စနစ်တကျ စီမံခန့်ခွဲသင့်သနည်း။ ယင်းတို့ကို သီးခြား Folder တစ်ခုထဲတွင် ထားရှိပြီး Version Control (Git) ဖြင့် ထိန်းသိမ်းကာ Script များကို အသုံးပြု၍ **Symlink** ချိတ်ဆက် ထားသင့်ပါသည်။ ဤသို့ ပြုလုပ်ခြင်းဖြင့် အောက်ပါ အကျိုးကျေးဇူးများ ရရှိနိုင်ပါသည်။

- **တပ်ဆင်ရ လွယ်ကူခြင်း**: စက်အသစ်တစ်လုံးသို့ ဝင်ရောက်သည့်အခါ သင်၏ စိတ်ကြိုက် Setting များကို တပ်ဆင်ရန် မိနစ်ပိုင်းမျှသာ ကြာမြင့်မည်။
- **သယ်ဆောင်ရ လွယ်ကူခြင်း**: သင်၏ Tool များသည် မည်သည့်နေရာတွင်မဆို တူညီစွာ အလုပ်လုပ်မည်။
- **ဟန်ချက်ညီ တူညီစေခြင်း**: မည်သည့်နေရာမှမဆို Dotfile များကို အပ်ဒိတ်လုပ်နိုင်ပြီး အားလုံးကို တူညီအောင် ထိန်းသိမ်းနိုင်မည်။
- **အပြောင်းအလဲများကို အမှတ်အသားပြုနိုင်ခြင်း**: Dotfile များကို သင်၏ ကွန်ပျူတာ သက်တမ်းတစ်လျှောက် ထိန်းသိမ်းသွားရမည် ဖြစ်ရာ Version History ရှိနေခြင်းသည် ရေရှည်အတွက် အလွန်ကောင်းမွန်ပါသည်။

Dotfile များထဲတွင် ဘာတွေ ထည့်သွင်းသင့်သနည်း။ အွန်လိုင်း Documentation များ သို့မဟုတ် **man pages** (https://en.wikipedia.org/wiki/Man_page) များကို ဖတ်ရှုခြင်းဖြင့် သင်၏ Tool Setting များကို လေ့လာနိုင်ပါသည်။ အခြား နည်းလမ်းတစ်ခုမှာ ပရိုဂရမ်တစ်ခုစီအတွက် ရေးသားထားသော Blog Post များကို ရှာဖွေဖတ်ရှုခြင်း ဖြစ်သည်။ ထို့အပြင် အခြားသူများ၏ Dotfile များကို ဝင်ရောက်ကြည့်ရှုခြင်းဖြင့်လည်း သင်ယူနိုင်ပါသည် - GitHub တွင် **Dotfiles Repositories** (https://github.com/search?o=desc&q=dotfiles&s=stars&type=Repositories) ပေါင်းများစွာကို ရှာဖွေတွေ့ရှိနိုင်ပါသည် — အသုံးအများဆုံး ရေပန်းစားလှသော repo ကို **ဒီမှာ** (https://github.com/mathiasbynens/dotfiles) ကြည့်ရှုနိုင်ပါသည် (သို့သော် ကူးယူဖတ်ရှုရာတွင် မစိစစ်ဘဲ ကူးယူခြင်း မပြုရန် အကြံပြုပါသည်)။ **ဤနေရာတွင်လည်း** (https://dotfiles.github.io/) အသုံးဝင်သော အချက်အလက်များကို လေ့လာနိုင်ပါသည်။

ဤသင်ခန်းစာ၏ ဆရာများ အားလုံးသည် ယင်းတို့၏ Dotfile များကို GitHub တွင် အများပြည်သူ ကြည့်ရှုနိုင်ရန် တင်ပေးထားကြပါသည် - **Anish** (https://github.com/anishathalye/dotfiles), **Jon** (https://github.com/jonhoo/configs), **Jose** (https://github.com/jjgo/dotfiles)။

Portability

Dotfile များ အသုံးပြုရာတွင် ကြုံတွေ့ရလေ့ရှိသော အခက်အခဲတစ်ခုမှာ စက်အများအပြားတွင် အလုပ်လုပ်သည့်အခါ (ဥပမာ- Operating System သို့မဟုတ် Shell ကွဲပြားနေခြင်း) Setting များ အဆင်မပြေဖြစ်ခြင်း ဖြစ်သည်။ အချို့သော Setting များကို သီးခြား စက်တစ်ခုထဲတွင်သာ အသုံးပြုချင်သည့် အခါမျိုးလည်း ရှိတတ်ပါသည်။

ဤကိစ္စအတွက် လွယ်ကူစေမည့် လှည့်ကွက်အချို့ ရှိပါသည်။ Configuration ဖိုင်က ထောက်ပံ့ပေးပါက သီးခြား စက်များအတွက် if-statement များနှင့် တူညီသော Logic များကို အသုံးပြုနိုင်ပါသည်။ ဥပမာ အားဖြင့် သင်၏ Shell တွင် အောက်ပါအတိုင်း ရေးသားနိုင်ပါသည်။

```
if [[ "$(uname)" == "Linux" ]]; then {do_something}; fi

# Shell သီးခြား အင်္ဂါရုပ်များကို မသုံးမီ စစ်ဆေးခြင်း
if [[ "$SHELL" == "zsh" ]]; then {do_something}; fi

# သီးခြား စက်အမည် (hostname) အတွက် စစ်ဆေးခြင်း
if [[ "$(hostname)" == "myServer" ]]; then {do_something}; fi
```

Configuration ဖိုင်က ထောက်ပံ့ပေးပါက include များကို အသုံးပြုပါ။ ဥပမာအားဖြင့် `~/.gitconfig` တွင် အောက်ပါ Setting ကို ထည့်သွင်းနိုင်ပါသည်။

```
[include]
    path = ~/.gitconfig_local
```

ထို့နောက် စက်တစ်ခုစီတွင် `~/.gitconfig_local` ဖိုင်ထဲ၌ ထိုစက်အတွက် သီးသန့် Setting များကို ထည့်သွင်းထားနိုင်ပါသည်။ ထို သီးသန့် Setting များကို Repository သီးခြား တစ်ခုဖြင့်ပင် ထိန်းသိမ်းထားနိုင်ပါသည်။

ဤအယူအဆသည် ပရိုဂရမ် ကွဲပြားသော်လည်း Setting အချို့ကို မျှဝေသုံးစွဲလိုသည့် အခါတွင်လည်း အသုံးဝင်ပါသည်။ ဥပမာအားဖြင့် `bash` နှင့် `zsh` နှစ်ခုစလုံးတွင် Alias စာရင်း တစ်ခုတည်းကို မျှဝေသုံးလိုပါက ယင်းတို့ကို `.aliases` ဖိုင်ထဲတွင် ရေးသားပြီး ဖိုင်နှစ်ခုစလုံး၌ အောက်ပါ Block ကို ထည့်သွင်းထားနိုင်ပါသည်။

```
# ~/.aliases ဖိုင် ရှိမရှိ စစ်ဆေးပြီး ရှိပါက လုဒ်ဆွဲယူပါ
if [ -f ~/.aliases ]; then
    source ~/.aliases
fi
```

Remote Machines

Programmer များအနေဖြင့် နေ့စဉ် အလုပ်ခွင်တွင် Remote Server များကို အသုံးပြုလာကြသည်မှာ ပိုမို များပြားလာခဲ့ပါသည်။ Backend Software များကို တပ်ဆင်ရန် သို့မဟုတ် ပိုမို မြင့်မားသော တွက်ချက်မှု စွမ်းရည် (computation capacity) လိုအပ်သည့်အခါမျိုးတွင် Remote Server များကို အသုံးပြုရန် Secure Shell (SSH) ကို အသုံးပြုရမည် ဖြစ်ပါသည်။ အခြားသော Tool များကဲ့သို့ပင် SSH သည် စိတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်း မြင့်မားသဖြင့် လေ့လာထားရန် အလွန် တန်ဖိုးရှိပါသည်။

Server တစ်ခုသို့ `ssh` ဝင်ရောက်ရန်အတွက် အောက်ပါအတိုင်း Command ကို runရပါမည် -

```
ssh foo@bar.mit.edu
```

ဤနေရာတွင် ကျွန်ုပ်တို့သည် `bar.mit.edu` ဟူသော Server အောက်ရှိ User `foo` အဖြစ် ssh ဝင်ရန် ကြိုးစားနေခြင်း ဖြစ်သည်။ Server အမည်ကို URL (ဥပမာ `bar.mit.edu`) သို့မဟုတ် IP address (ဥပမာ `foobar@192.168.1.42`) ဖြင့် သတ်မှတ်နိုင်ပါသည်။ နောက်ပိုင်းတွင် SSH Config ဖိုင်ကို ပြင်ဆင်လိုက်ပါက `ssh bar` ဟု ရိုက်ရုံဖြင့် ဝင်ရောက်နိုင်သည်ကို တွေ့မြင်ရပါမည်။

Executing commands

မကြာခဏ သတိမမူမိကြသည့် SSH ၏ အင်္ဂါရုပ်တစ်ခုမှာ Command များကို တိုက်ရိုက် runနိုင်သည့် စွမ်း ရည် ဖြစ်ပါသည်။ `ssh foobar@server ls` ဟု ရိုက်ပါက foobar ၏ home folder အတွင်း၌ `ls` ကို runသွားမည် ဖြစ်သည်။ ၎င်းသည် Pipe များနှင့်လည်း အလုပ်လုပ်နိုင်ပါသည်၊ ထို့ကြောင့် `ssh foobar@server ls | grep PATTERN` သည် Remote ဘက်မှ `ls` output ကို Local ဘက်တွင် `grep` စစ်ပေးမည် ဖြစ်ပြီး `ls | ssh foobar@server grep PATTERN` သည် Local ဘက်မှ output ကို Remote ဘက်သို့ ပေးပို့၍ `grep` စစ်ပေးမည် ဖြစ်သည်။

SSH Keys

Key ကို အခြေခံသော Authentication သည် Public-key Cryptography ကို အသုံးပြု၍ Server သို့ ဝင်ရောက် ရာတွင် Private Key ကို အပြင်သို့ ထုတ်မပြဘဲ Client ၌ သီးသန့် Private Key ရှိကြောင်း သက်သေပြ ပါသည်။ ဤနည်းဖြင့် လော့ဂ်အင် ဝင်တိုင်း စကားဝှက် (password) ကို ထပ်ခါတလဲလဲ ရိုက်နှိပ်စရာ မလို

တော့ပါ။ သို့သော် Private Key (အများအားဖြင့် `~/.ssh/id_rsa` သို့မဟုတ် နောက်ပိုင်းတွင် `~/.ssh/id_ed25519`) သည် သင်၏ Password ကဲ့သို့ပင် အရေးကြီးသဖြင့် သေချာစွာ ထိန်းသိမ်းပါ။

Key generation

Key အစုံ (pair) ကို ထုတ်ယူရန် **`ssh-keygen`** (https://www.man7.org/linux/man-pages/man1/ssh-keygen.1.html) ကို runနိုင်ပါသည်။

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

Private Key ကို အခြားသူများ ရရှိသွားပါက Server များကို မသမာဝင်ရောက်ခြင်းမှ ကာကွယ်ရန် Passphrase တစ်ခု သတ်မှတ်ထားသင့်ပါသည်။ Passphrase ကို အကြိမ်ကြိမ် ရိုက်မနေစေရန် **`ssh-agent`** (https://www.man7.org/linux/man-pages/man1/ssh-agent.1.html) သို့မဟုတ် **`gpg-agent`** (https://linux.die.net/man/1/gpg-agent) ကို အသုံးပြုနိုင်ပါသည်။

GitHub သို့ SSH Key အသုံးပြု၍ Push လုပ်ရန် ပြင်ဆင်ဖူးပါက [ဒီနေရာတွင်](https://help.github.com/articles/connecting-to-github-with-ssh/) (https://help.github.com/articles/connecting-to-github-with-ssh/) ဖော်ပြထားသော အဆင့်များကို လုပ်ဆောင်ဖူးမည် ဖြစ်ပြီး Key pair ရှိပြီးသား ဖြစ်နိုင်ပါသည်။ Passphrase ပါမပါ စစ်ဆေးရန်နှင့် စိစစ်ရန်အတွက် `ssh-keygen -y -f /path/to/key` ကို runနိုင်ပါသည်။

Key based authentication

`ssh` သည် မည်သည့် Client များကို ဝင်ရောက်ခွင့်ပြုရမည်ကို ဆုံးဖြတ်ရန် `.ssh/authorized_keys` ဖိုင်ကို စစ်ဆေးပါသည်။ Public Key ကို Remote စက်သို့ ကူးယူရန် အောက်ပါ Command ကို အသုံးပြုနိုင်ပါသည် -

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

သို့မဟုတ် အခြေခံ နည်းလမ်းအားဖြင့် -

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

Copying files over SSH

SSH မှတဆင့် ဖိုင်များကို ကူးယူနိုင်သည့် နည်းလမ်းများစွာ ရှိပါသည် -

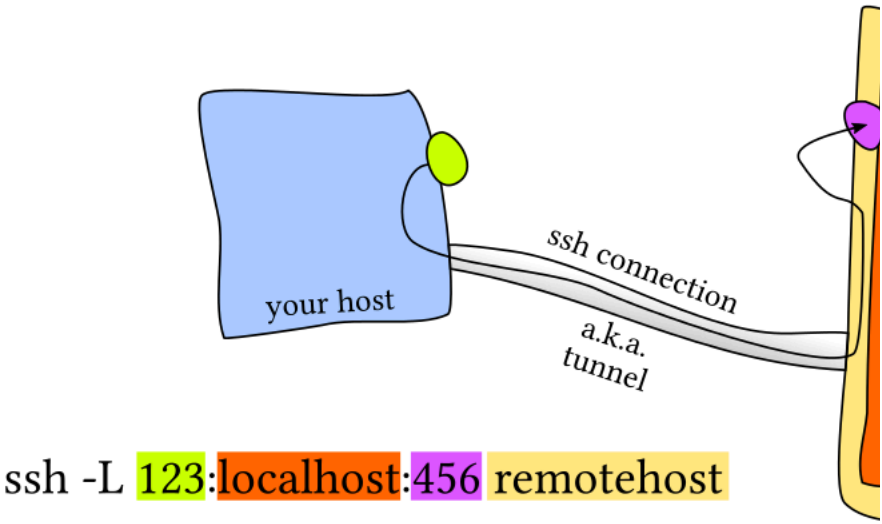
- `ssh+tee` - အလွယ်ကူဆုံးမှာ SSH command execution နှင့် STDIN input တို့ကို တွဲဖက်၍ `cat localfile | ssh remote_server tee serverfile` ဟု ပြုလုပ်ခြင်း ဖြစ်သည်။ **`tee`** (https://www.man7.org/linux/man-pages/man1/tee.1.html) သည် STDIN မှ လာသော Output များကို ဖိုင်ထဲသို့ ရေးသားပေးကြောင်း သတိရပါ။
- **`scp`** (https://www.man7.org/linux/man-pages/man1/scp.1.html) - ဖိုင် သို့မဟုတ် Directory အမြောက်အများကို ကူးယူသည့်အခါ Secure Copy `scp` Command သည် လမ်းကြောင်း အဆင့်ဆင့်ကို လွယ်ကူစွာ ကူးယူနိုင်သဖြင့် ပိုမို အဆင်ပြေပါသည်။ 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) - `rsync` သည် Local နှင့် Remote ကြား တူညီသော ဖိုင်များကို စစ်ဆေးတွေ့ရှိနိုင်ပြီး ထပ်မံ ကူးယူခြင်းကို တားဆီးပေးသဖြင့် `scp` ထက် ပိုမို ကောင်းမွန်ပါသည်။ ထို့အပြင် Symlinks၊ Permissions များကို ပိုမို အသေးစိတ် စီမံနိုင်ပြီး မပြီးသေးသော ကူးယူမှုများကို ပြန်လည် စတင်နိုင်သည့် `--partial` flag ကဲ့သို့သော အင်္ဂါရုပ်များ ပါဝင်သည်။ `rsync` ၏ Syntax မှာ `scp` နှင့် ဆင်တူပါသည်။

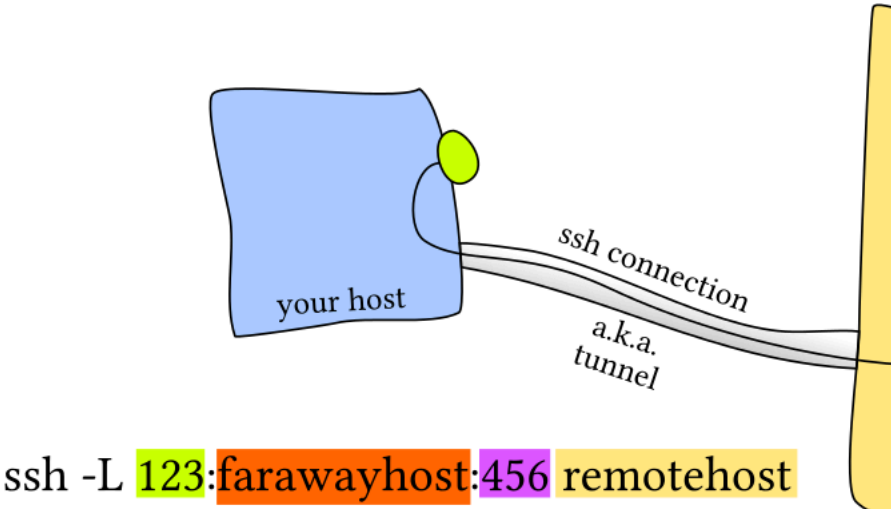
Port Forwarding

အခြေအနေများစွာတွင် စက်၏ သီးခြား Port များကို နားထောင်နေသော (listening) Software များနှင့် ကြုံတွေ့ရတတ်ပါသည်။ Local စက်တွင်ဆိုလျှင် `localhost:PORT` သို့မဟုတ် `127.0.0.1:PORT` ဟု ရိုက်နှိပ်နိုင်သော်လည်း Port များကို Network/Internet မှ တိုက်ရိုက် ဝင်ရောက်၍ မရသော Remote Server တွင် မည်သို့ ပြုလုပ်မည်နည်း။

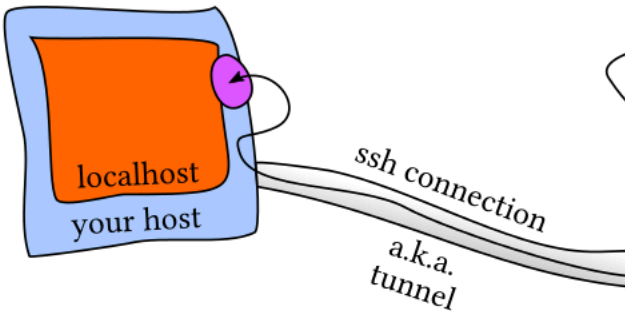
ဤသည်ကို *Port Forwarding* ဟု ခေါ်ဆိုပြီး မုဒ်နှစ်မျိုး ရှိပါသည် - Local Port Forwarding နှင့် Remote Port Forwarding (အသေးစိတ်ကို ပုံများတွင် ကြည့်ပါ၊ ပုံများ၏ မူပိုင်ခွင့်မှာ ဤ **StackOverflow Post** (https://unix.stackexchange.com/questions/115897/whats-ssh-port-forwarding-and-whats-the-difference-between-ssh-local-and-remot) မှ ဖြစ်ပါသည်)။

Local Port Forwarding

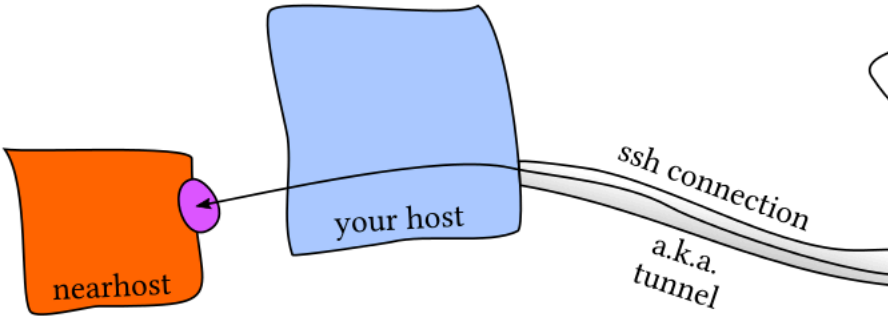

your host
ssh connection
a.k.a.
tunnel
ssh -L 123:localhost:456 remotehost


your host
ssh connection
a.k.a.
tunnel
ssh -L 123:farawayhost:456 remotehost

Remote Port Forwarding




```
ssh -R 123:localhost:456 remotehost
```




```
ssh -R 123:nearhost:456 remotehost
```

အတွေ့ရဆုံး အခြေအနေမှာ Remote စက်ရှိ Service ၏ Port ကို Local စက်ရှိ Port နှင့် ချိတ်ဆက်ပေးသော Local Port Forwarding ဖြစ်ပါသည်။ ဥပမာ Remote Server တွင် Port `8888` ကို နားထောင်နေသည့် `jupyter notebook` ကို runထားသည် ဆိုပါစို့။ ထို Port ကို Local Port `9999` သို့ Forward လုပ်ရန် အတွက် `ssh -L 9999:localhost:8888 foobar@remote_server` ဟု runပြီး Local စက် Browser တွင် `localhost:9999` သို့ ဝင်ရောက် ကြည့်ရှုနိုင်ပါသည်။

SSH Configuration

ကျွန်ုပ်တို့သည် ထည့်သွင်းနိုင်သော Argument များစွာကို လေ့လာခဲ့ပြီး ဖြစ်သည်။ ၎င်းအတွက် အလွယ်လမ်းမှာ အောက်ပါအတိုင်း Shell Alias များ ဖန်တီးခြင်း ဖြစ်နိုင်သည် -

```
alias my_server="ssh -i ~/.ssh/id_ed25519 --port 2222 -L 9999:localhost:8888 fo
obar@remote_server"
```

သို့သော် `~/.ssh/config` ဖိုင်ကို အသုံးပြုသည့် ပိုမို ကောင်းမွန်သော နည်းလမ်း ရှိပါသည်။

```
Host vm
    User foobar
    HostName 172.16.174.141
    Port 2222
    IdentityFile ~/.ssh/id_ed25519
    LocalForward 9999 localhost:8888

# Config များတွင် wildcard များကိုလည်း အသုံးပြုနိုင်သည်
Host *.mit.edu
    User foobaz
```

Alias များထက် `~/.ssh/config` ဖိုင်ကို အသုံးပြုခြင်း၏ အဓိက အားသာချက်မှာ `scp` , `rsync` , `mosh` အစရှိသော အခြား ပရိုဂရမ်များသည်လည်း ဤ Config ကို ဖတ်ရှုနိုင်ပြီး သက်ဆိုင်ရာ Flag များ အဖြစ် အလိုအလျောက် ပြောင်းလဲပေးနိုင်ခြင်း ဖြစ်ပါသည်။

သတိပြုရန်မှာ `~/.ssh/config` ဖိုင်ကို Dotfile တစ်ခုအဖြစ် သတ်မှတ်နိုင်ပြီး သင်၏ အခြား Dotfile များ နှင့်အတူ သိမ်းဆည်းနိုင်ပါသည်။ သို့သော် ယင်းကို အများပြည်သူသို့ ချပြလိုက်ပါက အင်တာနက်ပေါ်မှ အခြားသူများအား သင်၏ Server လိပ်စာများ၊ User အမည်များ၊ ဖွင့်ထားသော Port များ အစရှိသည်တို့

ပါဝင်သွားနိုင်ပါသည်။ ဤသည်မှာ တိုက်ခိုက်မှုအချို့ကို လွယ်ကူစေနိုင်သဖြင့် SSH Configuration များကို မျှဝေရာတွင် သတိထားပါ။

Server ဘက်ခြမ်း Configuration များကို အများအားဖြင့် `/etc/ssh/sshd_config` တွင် သတ်မှတ်လေ့ရှိသည်။ ဤနေရာတွင် Password authentication ကို ပိတ်ခြင်း၊ SSH Port များကို ပြောင်းလဲခြင်း၊ X11 forwarding ကို ဖွင့်ခြင်း အစရှိသည်တို့ကို ပြုလုပ်နိုင်ပါသည်။

Miscellaneous

Remote Server သို့ ချိတ်ဆက်ရာတွင် ကြုံတွေ့ရလေ့ရှိသော အခက်အခဲမှာ ကွန်ပျူတာ ပိတ်သွားခြင်း၊ Sleep ဝင်သွားခြင်း သို့မဟုတ် Network ပြောင်းသွားခြင်းတို့ကြောင့် ချိတ်ဆက်မှု ကျသွားခြင်း ဖြစ်သည်။ ထို့အပြင် Lag များသော လိုင်းဖြင့် SSH သုံးရခြင်းမှာ အလွန် စိတ်ပျက်ဖွယ် ကောင်းပါသည်။ **Mosh** (https://mosh.org/) (mobile shell) သည် SSH ကို ပို့မို့ ကောင်းမွန်အောင် ပြုလုပ်ထားပြီး Roaming ချိတ်ဆက်မှုများ၊ လိုင်းခေတ္တ ကျသွားမှုများကို ကြိုကြိုခံနိုင်ကာ လျင်မြန်သော Local Echo ကို ပေးစွမ်းနိုင်ပါသည်။

အချို့သော အခြေအနေများတွင် Remote Folder ကို Local တွင် Mount လုပ်ထားခြင်းက အဆင်ပြေပါသည်။ **sshfs** (https://github.com/libfuse/sshfs) သည် Remote Server ရှိ Folder ကို Local တွင် Mount လုပ်ပေးနိုင်ပြီး Local Editor ဖြင့် တိုက်ရိုက် ပြင်ဆင်နိုင်စေပါသည်။

Shells & Frameworks

Shell Tool နှင့် Scripting အပိုင်းတွင် အကျယ်ပြန့်ဆုံး သုံးစွဲကြသော `bash` shell အကြောင်းကို လေ့လာခဲ့ကြပါသည်။ သို့သော် ၎င်းတစ်ခုတည်းသာ ရွေးချယ်စရာ ရှိသည် မဟုတ်ပါ။

ဥပမာအားဖြင့် `zsh` shell သည် `bash` ၏ Superset ဖြစ်ပြီး မူလပါဝင်သော အဆင်ပြေသည့် အင်္ဂါရုပ်များစွာ ရှိပါသည် -

- ပိုမို စမတ်ကျသော Globbing, `**`
- Inline globbing/wildcard expansion
- စာလုံးပေါင်း အမှား ပြင်ဆင်ပေးခြင်း
- ပိုမိုကောင်းမွန်သော Tab completion/selection
- Path expansion (`cd /u/lo/b` သည် `/usr/local/bin` သို့ တိုးချဲ့သွားမည်)

Frameworks များသည်လည်း သင်၏ Shell ကို ပိုမိုကောင်းမွန်စေပါသည်။ ရေပန်းစားသော Framework အချို့မှာ **prezto** (https://github.com/sorin-ionescu/prezto) သို့မဟုတ် **oh-my-zsh** (https://ohmyz.sh/) တို့ ဖြစ်ကြပြီး၊ သီးခြား အင်္ဂါရုပ်များကို အာရုံစိုက်ထားသော **zsh-syntax-highlighting** (https://github.com/zsh-users/zsh-syntax-highlighting) သို့မဟုတ် **zsh-history-substring-search** (https://github.com/zsh-users/zsh-history-substring-search) တို့လည်း ရှိကြသည်။ **fish** (https://fishshell.com/) ကဲ့သို့သော Shell များတွင် ဤအင်္ဂါရုပ် အများအပြား မူလကတည်းက ပါဝင်ပြီးသား ဖြစ်သည်။

- Right prompt
- Command syntax highlighting
- History substring search
- manpage ကို အခြေခံသော flag completion များ
- စမတ်ကျသော autocompletion
- Prompt themes

ဤ Framework များကို အသုံးပြုရာတွင် သတိထားရန်မှာ ယင်းတို့သည် Shell ကို နှေးကွေးသွားစေနိုင်သည်ဟူသော အချက် ဖြစ်သည်။ လိုအပ်ပါက Profile လုပ်ကြည့်ပြီး မကြာခဏ မသုံးသော အင်္ဂါရုပ်များကို ပိတ်ထားနိုင်ပါသည်။

Terminal Emulators

သင်၏ Shell ကို ပြင်ဆင်သတ်မှတ်ခြင်းနှင့်အတူ သင်အသုံးပြုမည့် **Terminal Emulator** နှင့် ယင်း၏ Setting များကို ရွေးချယ်ရန် အချိန်ပေးသင့်ပါသည်။ Terminal Emulator အမျိုးအစားများစွာ ရှိပါသည် (ဒီမှာ (https://anarc.at/blog/2018-04-12-terminal-emulators-1/) နှိုင်းယှဉ်ချက်ကို ကြည့်နိုင်ပါသည်)။

Terminal တွင် နာရီပေါင်း ရာနှင့်ချီ၍ အချိန်ကုန်ဆုံးရမည် ဖြစ်ရာ Setting များကို လေ့လာ ပြင်ဆင်ခြင်းသည် အလွန် ထိရောက်သော ရင်းနှီးမြှုပ်နှံမှု ဖြစ်ပါသည်။ ပြင်ဆင်နိုင်သော အချက်အချို့မှာ -

- Font ရွေးချယ်မှု
- Color Scheme
- Keyboard shortcuts
- Tab/Pane ထောက်ပံ့မှု
- Scrollback configuration
- Performance (**Alacritty** (https://github.com/jwilm/alacritty) သို့မဟုတ် **kitty** (https://sw.kovidgoyal.net/kitty/) ကဲ့သို့ သော Terminal သစ်များသည် GPU Acceleration ကို ထောက်ပံ့ပေးသည်)

Exercises

Job control

1. `ps aux | grep` command များကို အသုံးပြု၍ Job PID များကို ရယူပြီး ပိတ်ပစ်နိုင်သော်လည်း ပိုမို ကောင်းမွန်သော နည်းလမ်းများ ရှိပါသည်။ Terminal တွင် `sleep 10000` ကို စတင် runပါ၊ `Ctrl-Z` ဖြင့် Background သို့ ပို့ပြီး `bg` ဖြင့် ဆက်လက် runပါ။ ယခု **`pgrep`** (https://www.man7.org/linux/man-pages/man1/pgrep.1.html) ကို အသုံးပြု၍ PID ကို ရှာပါ၊ ထို့နောက် PID ကို ကိုယ်တိုင် မရိုက်ရဘဲ **`pkill`** (https://man7.org/linux/man-pages/man1/pgrep.1.html) ဖြင့် ပိတ်ပစ်ပါ။ (အချက်အလက်- `-af` flag များကို အသုံးပြုပါ)။

2. Process တစ်ခု မပြီးမချင်း နောက် Process တစ်ခုကို မစတင်ချင်ဟု ဆိုပါစို့။ မည်သို့ ပြုလုပ်မည်နည်း။ ဤ လေ့ကျင့်ခန်းတွင် `sleep 60 &` ကို စောင့်ဆိုင်းမည့် Process အဖြစ် အသုံးပြုပါမည်။ အသုံးပြုနိုင်သော နည်းလမ်းတစ်ခုမှာ **`wait`** (https://www.man7.org/linux/man-pages/man1/wait.1p.html) Command ကို အသုံးပြုခြင်း ဖြစ်သည်။ sleep command ကို runပြီး background process မပြီးမချင်း `ls` က စောင့်ဆိုင်းနေအောင် ပြုလုပ်ပါ။

သို့သော် ဤနည်းလမ်းသည် အခြား bash session မှ စတင်ပါက အဆင်မပြေပါ၊ အကြောင်းမှာ `wait` သည် child process များအတွက်သာ အလုပ်လုပ်သောကြောင့် ဖြစ်သည်။ `kill` command ၏ exit status သည် အောင်မြင်ပါက zero ဖြစ်ပြီး မအောင်မြင်ပါက nonzero ဖြစ်ပါသည်။ `kill -0` သည် signal မပို့သော်လည်း process မရှိပါက nonzero exit status ပေးမည် ဖြစ်သည်။
PID တစ်ခုကို လက်ခံပြီး ထို process မပြီးမချင်း စောင့်ဆိုင်းပေးမည့် `pidwait` ဟုခေါ်သော bash function တစ်ခု ရေးပါ။ CPU အလဟဿ မဖြစ်စေရန် `sleep` ကို အသုံးပြုပါ။

Terminal multiplexer

1. ဤ `tmux` [သင်ခန်းစာ](https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) ကို လေ့လာပြီး ဤ [အဆင့်များ](https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/) အတိုင်း စိတ်ကြိုက် ပြင်ဆင်နည်းကို လေ့လာပါ။

Aliases

1. စာလုံးပေါင်း မှားရိုက်မိသည့်အခါ `cd` သို့ ရောက်ရှိသွားစေရန် `dc` ဟူသော Alias တစ်ခု ဖန်တီးပါ။

2. `history | awk '{$1="";print substr($0,2)}' | sort | uniq -c | sort -n | tail -n 10` ကို run၍ အသုံးအများဆုံး Command ၁၀ ခုကို ရယူပြီး ယင်းတို့အတွက် အမည်တို Alias များ ရေးသားပါ။ (သတိပြုရန်- ဤသည်မှာ Bash အတွက် ဖြစ်သည်၊ ZSH သုံးပါက `history` အစား `history 1` ဟု သုံးပါ)။

Dotfiles

Dotfiles များ ပြင်ဆင်ကြပါစို့။ 1. သင်၏ Dotfiles အတွက် Folder တစ်ခု ဖန်တီးပြီး Version Control (Git) ပြုလုပ်ပါ။ 1. ပရိုဂရမ် အနည်းဆုံး တစ်ခု (ဥပမာ- သင်၏ Shell) အတွက် Customization ဖိုင် ထည့်သွင်းပါ (စတင်ရန်အတွက် `$PS1` ကို သတ်မှတ်၍ Prompt ပြင်ဆင်ခြင်းကဲ့သို့ လွယ်ကူသော အရာ ဖြစ်နိုင်ပါသည်။) 1. စက်အသစ်တွင် Dotfiles များကို မြန်ဆန်စွာ တပ်ဆင်နိုင်မည့် နည်းလမ်းကို ပြင်ဆင်ပါ။ (ဖိုင်တစ်ခုစီ အတွက် `ln -s` ခေါ်ပေးသည့် Shell script သို့မဟုတ် [အထူးပြု Utility](https://dotfiles.github.io/utilities/) ကို အသုံးပြုနိုင်သည်)။ 1. သင်၏ တပ်ဆင်မှု Script ကို Virtual Machine အသစ်တစ်ခုတွင် စမ်းသပ်ပါ။ 1. လက်ရှိ Tool Configuration အားလုံးကို သင်၏ Dotfiles repository သို့ ပြောင်းရွှေ့ပါ။ 1. သင်၏ Dotfiles များကို GitHub တွင် လွှင့်တင်ပါ။

Remote Machines

ဤလေ့ကျင့်ခန်းအတွက် Linux Virtual Machine တစ်ခု တပ်ဆင်ပါ (သို့မဟုတ် ရှိပြီးသားကို သုံးပါ)။ VM များ အကြောင်း မသိပါက [ဤသင်ခန်းစာ](https://hibbard.eu/install-ubuntu-virtual-box/) ကို ကြည့်ပါ။

1. `~/.ssh/` သို့ သွားပြီး SSH key pair ရှိမရှိ စစ်ဆေးပါ။ မရှိပါက `ssh-keygen -a 100 -t ed25519` ဖြင့် ထုတ်ယူပါ။ Password နှင့် `ssh-agent` အသုံးပြုရန် အကြံပြုပါသည် ([ဒီမှာ](https://www.ssh.com/ssh/agent) ပိုမို ကြည့်နိုင်ပါသည်)။

2. `.ssh/config` တွင် အောက်ပါအတိုင်း ထည့်သွင်းပါ -

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

3. `ssh-copy-id vm` ကို အသုံးပြု၍ Server သို့ SSH key ကူးယူပါ။

4. VM တွင် `python -m http.server 8888` ကို run၍ Web server စတင်ပါ။ မိမိစက်မှ `http://localhost:9999` သို့ ဝင်ရောက်၍ VM Web server ကို ကြည့်ပါ။

5. `sudo vim /etc/ssh/sshd_config` ကို ပြုလုပ်၍ `PasswordAuthentication` တန်ဖိုးကို ပြောင်းလဲကာ Password Authentication ကို ပိတ်ပါ။ `PermitRootLogin` တန်ဖိုးကို ပြောင်း၍ Root Login ကို ပိတ်ပါ။ `sudo service sshd restart` ဖြင့် SSH Service ကို ပြန်စပါ။ SSH ပြန်ဝင်ကြည့်ပါ။

6. (စိန်ခေါ်မှု) VM တွင် **`mosh`** (https://mosh.org/) တပ်ဆင်ပြီး ချိတ်ဆက်ပါ။ ထို့နောက် Server/VM ၏ Network adapter ကို ဖြုတ်လိုက်ပါ။ Mosh သည် ယင်းမှ အဆင်ပြေစွာ ပြန်လည် ချိတ်ဆက်နိုင်ပါသလား။

7. (စိန်ခေါ်မှု) SSH တွင် `-N` နှင့် `-f` flag များ အလုပ်လုပ်ပုံကို လေ့လာပြီး Background Port Forwarding ပြုလုပ်နိုင်သော Command ကို ရှာဖွေပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

```
</tbody>
```

Resource / Description	URL	Micro QR
1. `kill`	https://www.man7.org/linux/man-pages/man1/kill.1.html	
2. `fg`	https://www.man7.org/linux/man-pages/man1/fg.1p.html	
3. `bg`	https://man7.org/linux/man-pages/man1/bg.1p.html	
4. `jobs`	https://www.man7.org/linux/man-pages/man1/jobs.1p.html	
5. `pgrep`	https://www.man7.org/linux/man-pages/man1/pgrep.1.html	
6. `nohup`	https://www.man7.org/linux/man-pages/man1/nohup.1.html	
7. ဒီမှာ	https://en.wikipedia.org/wiki/Signal_(IPC	
8. `man signal`	https://www.man7.org/linux/man-pages/man7/signal.7.html	
9. `tmux`	https://www.man7.org/linux/man-pages/man1/tmux.1.html	
10. ` သည် Scrollback mode သို့ ဝင်မည်။ ထို့နောက် `` နှိပ်၍ highlight လုပ်ပြီး `` နှိပ်၍ copy ကူးနိုင်သည်။ + `` သည် Pane ၏ အခင်းအကျင်း ပုံစံများကို အစဉ်လိုက် ပြောင်းလဲပေးမည်။ ပိုမို လေ့လာလိုပါက၊ `tmux` အကြောင်း အတိုချုပ် သင်ခန်းစာ	https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/	
11. ဤသင်ခန်းစာ	https://linuxcommand.org/lc3_adv_termmux.php	
12. `screen`	https://www.man7.org/linux/man-pages/man1/screen.1.html	

Resource / Description	URL	Micro QR
13. `alias`	https://www.man7.org/linux/man-pages/man1/alias.1p.html	
14. ဒီမှာ	https://web.archive.org/web/20260329133158/https://blog.flowblok.id.au/2013-02/shell-startup-scripts.html	
15. man pages	https://en.wikipedia.org/wiki/Man_page	
16. Dotfiles Repositories	https://github.com/search?o=desc&q=dotfiles&s=stars&type=Repositories	
17. ဒီမှာ	https://github.com/mathiasbynens/dotfiles	
18. ဤနေရာတွင်လည်း	https://dotfiles.github.io/	
19. Anish	https://github.com/anishathalye/dotfiles	
20. Jon	https://github.com/jonhoo/configs	
21. Jose	https://github.com/jjgo/dotfiles	
22. `ssh-keygen`	https://www.man7.org/linux/man-pages/man1/ssh-keygen.1.html	
23. `ssh-agent`	https://www.man7.org/linux/man-pages/man1/ssh-agent.1.html	
24. `gpg-agent`	https://linux.die.net/man/1/gpg-agent	
25. ဒီနေရာတွင်	https://help.github.com/articles/connecting-to-github-with-ssh/	
26. `tee`	https://www.man7.org/linux/man-pages/man1/tee.1.html	
27. `scp`	https://www.man7.org/linux/man-pages/man1/scp.1.html	
28. `rsync`	https://www.man7.org/linux/man-pages/man1/rsync.1.html	

Resource / Description	URL	Micro QR
29. ဤ StackOverflow Post	https://unix.stackexchange.com/questions/115897/whats-ssh-port-forwarding-and-whats-the-difference-between-ssh-local-and-remot	
30. Mosh	https://mosh.org/	
31. sshfs	https://github.com/libfuse/sshfs	
32. prezto	https://github.com/sorin-ionescu/prezto	
33. oh-my-zsh	https://ohmyz.sh/	
34. zsh-syntax-highlighting	https://github.com/zsh-users/zsh-syntax-highlighting	
35. zsh-history-substring-search	https://github.com/zsh-users/zsh-history-substring-search	
36. fish	https://fishshell.com/	
37. ဒီမှာ	https://anarc.at/blog/2018-04-12-terminal-emulators-1/	
38. Alacritty	https://github.com/jwilm/alacritty	
39. kitty	https://sw.kovidgoyal.net/kitty/	
40. `pkill`	https://man7.org/linux/man-pages/man1/pgrep.1.html	
41. `wait`	https://www.man7.org/linux/man-pages/man1/wait.1p.html	
42. ဤအဆင့်များ	https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/	
43. အထူးပြု Utility	https://dotfiles.github.io/utilities/	
44. ဤသင်ခန်းစာ	https://hibbard.eu/install-ubuntu-virtual-box/	

Resource / Description	URL	Micro QR
45. ဒီမှာ	https://www.ssh.com/ssh/agent	

```
\pagebreak

# Course Overview + The Shell

> **အနှစ်ချုပ်** - ဤသင်တန်းကို သင်ကြားပေးရသည့် ရည်ရွယ်ချက်နှင့် Shell ကို စတင် အသုံးပြုနည်းကို လေ့လာပါ။


```{=html}
<div class="video-card">
  <div class="video-card-header">
    <span class="video-badge">▶ LECTURE VIDEO REFERENCE</span>
    <h3 class="video-title">Course Overview + The Shell</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_Z56Jmr9Z34Q.jpg" alt="Course O
verview + The Shell 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_Z56Jmr9Z34Q.png" alt="QR Code f
or Course Overview + The Shell" 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=Z56Jmr9Z34Q" class="video-ur
l">https://www.youtube.com/watch?v=Z56Jmr9Z34Q</a>
        </div>
      </div>
    </div>
  </div>
</div>
```
```

စိတ်ဓာတ်တက်ကြွမှုနှင့် ရည်ရွယ်ချက် (Motivation)

ကွန်ပျူတာသိပ္ပံပညာရှင်များအနေဖြင့် ကွန်ပျူတာများသည် ထပ်ခါတလဲလဲ ပြုလုပ်ရသည့် အလုပ်များကို အလိုအလျောက် ကူညီဆောင်ရွက်ပေးရာတွင် အလွန်တော်စွမ်းကြောင်း ကျွန်ုပ်တို့ သိကြပါသည်။ သို့သော် ဤအချက်သည် ကျွန်ုပ်တို့ ရေးသားသော ပရိုဂရမ်များ၏ တွက်ချက်မှုများအတွက်သာမက၊ မိမိတို့ ကိုယ်တိုင် ကွန်ပျူတာကို *အသုံးပြုမှု* တွင်ပါ တူညီစွာ သက်ရောက်ကြောင်း မကြာခဏ မေ့လျော့နေတတ်ကြသည်။ ကွန်ပျူတာနှင့် သက်ဆိုင်သော မည်သည့် ပြဿနာကိုမဆို ဖြေရှင်းရာတွင် ပိုမို အလုပ်တွင်စေပြီး ပိုမို ရှုပ်ထွေးသော ပြဿနာများကို ဖြေရှင်းနိုင်စေမည့် tool အမျိုးအစားများစွာကို ကျွန်ုပ်တို့ လက်တစ်ကမ်းတွင် ရရှိနိုင်ပါသည်။ သို့သော် ကျွန်ုပ်တို့အနက် အများစုသည် ယင်း tool များ၏ အစိတ်အပိုင်း အနည်းငယ်မျှကိုသာ အသုံးပြုကြပြီး၊ အခက်အခဲ ကြုံတွေ့ရသည့်အခါ အင်တာနက်မှ command များကို အလွတ်ကျက်၍ ကူးယူကူးထည့် (copy-paste) ပြုလုပ်ရုံမျှသာ တတ်မြောက်ထားကြသည်။

ဤသင်တန်းသည် ဤပြဿနာကို ဖြေရှင်းရန် ကြိုးပမ်းချက် ဖြစ်ပါသည်။

သင် သိရှိပြီးသား tool များကို အထိရောက်ဆုံး မည်သို့ အသုံးချရမည်၊ မိမိ၏ tool အိတ်အတွင်းသို့ tool အသစ်များ မည်သို့ ထည့်သွင်းရမည်ကို သင်ကြားပေးလိုပြီး၊ မိမိကိုယ်တိုင် tool များကို ပိုမို လေ့လာရှာဖွေရန် (နှင့် ဖန်တီးရန်) စိတ်အားထက်သန်မှုများ ရရှိစေရန် မျှော်လင့်ပါသည်။ ဤသည်မှာ ကွန်ပျူတာသိပ္ပံ သင်ရိုးညွှန်းတမ်း အများစုတွင် လိုအပ်နေသော သင်ရိုးဖြစ်သည်ဟု ကျွန်ုပ်တို့ ယုံကြည်ပါသည်။

သင်တန်း ဖွဲ့စည်းပုံ (Class structure)

ဤသင်တန်းတွင် ၁ နာရီကြာ သင်ခန်းစာ ၁၁ ခု ပါဝင်ပြီး သင်ခန်းစာ တစ်ခုစီသည် သီးခြား ခေါင်းစဉ်တစ်ခု (https://missing-semester-my.github.io/2020/) ပေါ်တွင် ဗဟိုပြုထားပါသည်။ သင်ခန်းစာတစ်ခုချင်းစီသည် သီးခြားစီ လေ့လာနိုင်သော ခေါင်းစဉ်များ ဖြစ်ကြသော်လည်း၊ သင်တန်းကာလ တိုးတက်လာသည်နှင့်အမျှ ယခင် သင်ခန်းစာပါ အကြောင်းအရာများကို သိရှိပြီးဖြစ်သည်ဟု ယူဆသွားမည် ဖြစ်ပါသည်။ သင်ခန်းစာ မှတ်စု များကို အွန်လိုင်းတွင် ရယူနိုင်သော်လည်း၊ သင်ခန်းစာ မှတ်စုတွင် ပါဝင်မည်မဟုတ်သည့် လက်တွေ့ပြသမှု (demo) များစွာကို အတန်းထဲတွင် သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။ သင်ခန်းစာ ရိုက်ကူးချက်များကို လည်း အွန်လိုင်းတွင် တင်ပေးထားမည် ဖြစ်ပါသည်။

၁ နာရီကြာ သင်ခန်းစာ ၁၁ ခုအတွင်း အကြောင်းအရာများစွာကို လွှမ်းခြုံနိုင်ရန် ကြိုးစားထားသဖြင့် သင်ခန်းစာများသည် အလွန် သိပ်သည်းပါသည်။ မိမိ၏ လေ့လာမှု အရှိန်အဟုန်အတိုင်း လေ့လာနိုင်ရန် အတွက် သင်ခန်းစာ တစ်ခုစီတွင် အဓိက အချက်များကို လမ်းညွှန်ပေးသော လေ့ကျင့်ခန်းများ ပါဝင်ပါသည်။ သင်ခန်းစာများအပြီးတွင် မေးမြန်းလိုသည်များအတွက် Office Hours ပြုလုပ်ပေးသွားမည် ဖြစ်ပါသည်။ အွန် လိုင်းမှ တက်ရောက်ပါက မေးခွန်းများကို missing-semester@mit.edu သို့ ပို့ဆိုနိုင်ပါသည်။

အချိန် အကန့်အသတ်ရှိသဖြင့် အပြည့်အစုံ သင်ကြားပေးသော သင်တန်းတစ်ခုကဲ့သို့ too! အားလုံးကို အသေးစိတ် မသင်ကြားနိုင်ပါ။ တတ်နိုင်သမျှ သက်ဆိုင်ရာ too! သို့မဟုတ် ခေါင်းစဉ်ကို ပိုမို နက်နက်နဲနဲ လေ့လာနိုင်မည့် အရင်းအမြစ်များကို ညွှန်းဆိုပေးသွားမည် ဖြစ်ပါသည်။

ခေါင်းစဉ် ၁: Shell (Topic 1: The Shell)

Shell ဆိုတာ ဘာလဲ? (What is the shell?)

ယနေ့ခေတ် ကွန်ပျူတာများတွင် Command ပေးနိုင်သော Interface အမျိုးမျိုး ရှိကြပါသည်—လှပသော Graphical User Interface (GUI) များ၊ Voice Interface များ၊ နှင့် AR/VR များအထိ နေရာတိုင်းတွင် ရှိနေကြသည်။ ၎င်းတို့သည် သုံးစွဲမှု ၈၀% အတွက် အလွန် ကောင်းမွန်သော်လည်း၊ ပြုလုပ်နိုင်သည့် အတိုင်းအတာတွင် မူလကတည်းက ကန့်သတ်ချက်များ ရှိနေသည်—မရှိသေးသော ခလုတ်တစ်ခုကို နှိပ်၍ မရသကဲ့သို့ ပရိုဂရမ် မလုပ်ရသေးသော အသံမိန်ခွန်းပေးချက်ကို ပေး၍ မရပါ။ မိမိ ကွန်ပျူတာက ပံ့ပိုးပေးထားသော tool များကို အပြည့်အဝ အသုံးချနိုင်ရန်အတွက် ရှေးမူလ စာသား အခြေပြု Interface ဖြစ်သည့် **The Shell** ထံသို့ ဆင်းသက် အသုံးပြုရမည် ဖြစ်သည်။

သင် အသုံးပြုနိုင်သည့် စက်အများစုတွင် Shell တစ်မျိုးမဟုတ် တစ်မျိုး ပါဝင်ပြီး၊ အချို့တွင် ရွေးချယ်စရာ Shell အမျိုးမျိုး ပါရှိကြသည်။ အသေးစိတ် ကွဲပြားနိုင်သော်လည်း၊ အခြေခံအားဖြင့် ၎င်းတို့သည် တူညီကြသည်—ပရိုဂရမ်များကို Run ရန်၊ Output များကို ကြည့်ရှုရန်နှင့် စနစ်တကျ အလုပ်လုပ်ရန် အခွင့်အရေး ပေးထားသည်။

ဤသင်ခန်းစာတွင် Bourne Again SHell သို့မဟုတ် အတိုကောက် **“bash”** ကို အဓိက ထား သင်ကြားပေးသွားမည် ဖြစ်သည်။ ဤသည်မှာ အသုံးအများဆုံး Shell များအနက် တစ်ခုဖြစ်ပြီး ယင်း၏ Syntax သည် အခြား Shell အများစုနှင့် ဆင်တူပါသည်။ Command များကို ရိုက်ထည့်နိုင်သော Shell Prompt ဖွင့်လှစ်ရန် အတွက် ပထမဦးစွာ **Terminal** တစ်ခု လိုအပ်ပါသည်။ သင်၏ စက်တွင် Terminal ပါဝင်ပြီးဖြစ်ပါလိမ့်မည်။

Shell ကို အသုံးပြုခြင်း (Using the shell)

Terminal ကို ဖွင့်လှစ်လိုက်သည့်အခါ အောက်ပါအတိုင်း တွေ့မြင်ရမည့် **Prompt** ကို တွေ့ရမည် ဖြစ်သည်-

```
missing:~$
```

ဤသည်မှာ Shell ၏ အဓိက စာသား Interface ဖြစ်သည်။ သင်သည် `missing` ဆိုသည့် စက်ပေါ်တွင် ရောက်ရှိနေပြီး မိမိ၏ “Current Working Directory” (လက်ရှိ ရောက်ရှိနေသော Directory) မှာ `~` (Home ၏ အတိုကောက်) ဖြစ်ကြောင်း ဖော်ပြနေခြင်း ဖြစ်သည်။ `$` သင်္ကေတသည် သင်သည် Root user မဟုတ်

ကြောင်း ဖော်ပြနေခြင်း ဖြစ်သည်။ ဤ Prompt တွင် Shell မှ အဓိပ္ပာယ်ဖော်ယူမည့် Command ကို ရိုက်ထည့်နိုင်ပါသည်။ အခြေခံအကျဆုံး Command မှာ ပရိုဂရမ်တစ်ခုကို Run ခြင်း ဖြစ်သည်-

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

ဒီနေရာမှာ `date` ပရိုဂရမ်ကို Execute လုပ်ခဲ့ပြီး၊ ယင်းက လက်ရှိ ရက်စွဲနှင့် အချိန်ကို ရိုက်နှိပ်ပြသခဲ့သည်။ ထို့နောက် Shell က အခြား Execute လုပ်မည့် Command ကို ထပ်မံ တောင်းဆိုသည်။ Arguments (ကိန်းရှင်/အရာဝတ္ထု) များနှင့်လည်း Command ကို Execute လုပ်နိုင်သည်-

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

ဤနေရာတွင် `echo` ပရိုဂရမ်အား `hello` ဆိုသည့် Argument ဖြင့် Run ရန် Shell ကို ခိုင်းစေခဲ့ခြင်း ဖြစ်သည်။ `echo` ပရိုဂရမ်သည် ၎င်းထံ ပို့လိုက်သော Argument ကို ပြန်လည် ထုတ်ပေးရုံမျှသာ ပြုလုပ်သည်။ Shell သည် Command ကို ဟာကွက် (whitespace) များဖြင့် ခွဲခြား၍ အဓိပ္ပာယ်ဖော်ပြီး ပထမဆုံး စကားလုံးအတိုင်း ပရိုဂရမ်ကို Run ၍ နောက်ဆက်တွဲ စကားလုံးများကို Argument အဖြစ် ပေးပို့သည်။ ဟာကွက် သို့မဟုတ် အထူး သင်္ကေတများ ပါဝင်သော Argument (ဥပမာ “My Photos” အမည်ရှိ Directory) ပေးပို့လိုပါက `'` သို့မဟုတ် `"` ဖြင့် အုပ်ပေးနိုင်သည် (`"My Photos"`) သို့မဟုတ် သင်္ကေတ ရှေ့တွင် `\` ခံပေးနိုင်သည် (`My\ Photos`)။

သို့သော် Shell သည် `date` သို့မဟုတ် `echo` ပရိုဂရမ်များကို မည်သို့ ရှာဖွေရမည်ကို မည်သို့ သိရှိသနည်း။ Shell သည် Python သို့မဟုတ် Ruby ကဲ့သို့သော ပရိုဂရမ်းမင်း ပတ်ဝန်းကျင်တစ်ခု ဖြစ်သဖြင့် Variables များ၊ Conditionals များ၊ Loops များနှင့် Functions များ ပါဝင်ကြသည်။ Shell တွင် Command များ Run သည့်အခါ Shell က အဓိပ္ပာယ်ဖော်မည့် ကုဒ် အသေးစားလေးကို ရေးသားနေခြင်း ဖြစ်သည်။ Command တစ်ခုသည် သီးသန့် Keyword များနှင့် မကိုက်ညီပါက၊ Shell စက်ပေါ်တွင် ပရိုဂရမ်များ ရှာဖွေရမည့် Directory စာရင်း ပါဝင်သော `$PATH` ဆိုသည့် Environment Variable ကို စစ်ဆေးပါသည်-

```
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
```

`echo` command ကို Run သည့်အခါ Shell က `$PATH` ထဲရှိ `:` ဖြင့် ခွဲခြားထားသော Directory စာရင်းများအတွင်း ထိုအမည်ရှိ ဖိုင်ကို ရှာဖွေသည်။ တွေ့ရှိပါက Run ပေးသည် (ဖိုင်သည် Executable ဖြစ်ပါက)။ သတ်မှတ်ထားသော ပရိုဂရမ် မည်သည့် ဖိုင်မှ Execute ဖြစ်သည်ကို `which` ပရိုဂရမ် ဖြင့် ရှာဖွေနိုင်ပါသည်။ `$PATH` ကို ကျော်လွန်၍ ဖိုင်၏ လမ်းကြောင်း (path) အပြည့်အစုံ ပေး၍လည်း Run နိုင်ပါသည်။

Shell အတွင်း လမ်းကြောင်းရှာခြင်း (Navigating in the shell)

Shell ပေါ်ရှိ Path သည် Directory များ၏ လမ်းကြောင်းဖြစ်ပြီး Linux/macOS တွင် `/` ဖြင့်လည်းကောင်း Windows တွင် `\` ဖြင့်လည်းကောင်း ခွဲခြားထားသည်။ Linux နှင့် macOS တွင် `/` လမ်းကြောင်းသည် File System ၏ “Root” (ပင်မ) ဖြစ်ပြီး Directory နှင့် File အားလုံး ယင်းအောက်တွင် ရှိကြသည်။ `/` ဖြင့် စတင်သော Path ကို **Absolute path** ဟု ခေါ်ဆိုပြီး အခြား Path များကို **Relative path** ဟု ခေါ်ဆိုသည်။ Relative path သည် လက်ရှိ ရောက်ရှိနေသော Directory ပေါ်မူတည်ပြီး `pwd` command ဖြင့် ကြည့်ရှုနိုင်ကာ `cd` command ဖြင့် ပြောင်းလဲနိုင်ပါသည်။ Path တွင် `.` သည် လက်ရှိ Directory ကို ညွှန်းဆိုပြီး `..` သည် မိခင် (Parent) Directory ကို ညွှန်းဆိုသည်-

```
missing:~$ pwd
/home/missing
missing:~$ cd /home
missing:/home$ pwd
/home
missing:/home$ cd ..
missing:/$ pwd
/
missing:/$ cd ./home
missing:/home$ pwd
/home
missing:/home$ cd missing
missing:~$ pwd
/home/missing
missing:~$ ../../bin/echo hello
hello
```

မိမိ ရောက်ရှိနေသော Directory အတွင်း မည်သည့် ဖိုင်/ဖိုင်တွဲများ ရှိသည်ကို ကြည့်ရှုရန် `ls` command ကို အသုံးပြုသည်-

```
missing:~$ ls
missing:~$ cd ..
missing:/home$ ls
missing
missing:/home$ cd ..
missing:/$ ls
bin
boot
dev
etc
home
...
```

Command အများစုသည် ၎င်းတို့၏ လုပ်ဆောင်ချက်ကို ပြောင်းလဲရန် `-` ဖြင့် စတင်သော Flags နှင့် Options များကို လက်ခံကြသည်။ ပုံမှန်အားဖြင့် `-h` သို့မဟုတ် `--help` flag ဖြင့် Run ပါက အကူအညီ စာသားများကို ရိုက်နှိပ်ပြသပေးသည်။ ဥပမာ `ls --help` က အောက်ပါအတိုင်း ပြသသည်-

```
  -l                         use a long listing format
```

```
missing:~$ ls -l /home
drwxr-xr-x 1 missing  users  4096 Jun 15  2019 missing
```

ဤသည်မှာ ဖိုင် သို့မဟုတ် Directory ၏ အသေးစိတ် အချက်အလက်များကို ဖော်ပြပေးခြင်း ဖြစ်သည်။ ပထမဆုံး စာကြောင်းစအစရှိ `d` သည် `missing` မှာ Directory ဖြစ်ကြောင်း ဖော်ပြသည်။ ထို့နောက် စာလုံး ၃ လုံးပါ အုပ်စု ၃ စု (`rwx`) ပါရှိပြီး ယင်းတို့သည် ပိုင်ရှင် (`missing`)၊ ပိုင်ဆိုင်သော အုပ်စု (`users`)၊ နှင့် အခြားသူများ၏ ရပိုင်ခွင့် ခွင့်ပြုချက် (Permissions) များကို အစဉ်လိုက် ဖော်ပြထားခြင်း ဖြစ်သည်။ `r` မှာ Read၊ `w` မှာ Write (ပြင်ဆင်ခြင်း)၊ `x` မှာ Execute (Run ခြင်း သို့မဟုတ် Directory ထဲ ဝင်ရောက်ခြင်း) ဖြစ်ကြသည်။

အခြား အသုံးဝင်သော Command များမှာ `mv` (ဖိုင် အမည်ပြောင်း/ရွှေ့ခြင်း)၊ `cp` (ဖိုင် ကူးယူခြင်း)၊ နှင့် `mkdir` (Directory အသစ် ဖန်တီးခြင်း) တို့ ဖြစ်ကြသည်။

ပရိုဂရမ်တစ်ခု၏ အသေးစိတ် မူရင်း လမ်းညွှန်ချက် စာအုပ်ကို ကြည့်လိုပါက `man` command ကို အသုံးပြုနိုင်ပြီး ထွက်ခွာရန် `q` ကို နှိပ်ပါ-

```
missing:~$ man ls
```

ပရိုဂရမ်များကို ချိတ်ဆက်ခြင်း (Connecting programs)

Shell တွင် ပရိုဂရမ်များနှင့် သက်ဆိုင်သော အဓိက “Streams” နှစ်ခု ရှိပါသည်—ယင်းတို့မှာ Input stream နှင့် Output stream တို့ ဖြစ်ကြသည်။ ပုံမှန်အားဖြင့် Input မှာ Keyboard ဖြစ်ပြီး Output မှာ သင်၏ ဖန်သားပြင် ဖြစ်သည်။ သို့သော် ဤ Stream များကို လမ်းကြောင်း ပြောင်းလဲပေးနိုင်ပါသည်!

အလွယ်ကူဆုံး လမ်းကြောင်း ပြောင်းလဲခြင်း (Redirection) မှာ `< file` နှင့် `> file` တို့ ဖြစ်ကြသည်-

```
missing:~$ echo hello > hello.txt
missing:~$ cat hello.txt
hello
missing:~$ cat < hello.txt
hello
missing:~$ cat < hello.txt > hello2.txt
missing:~$ cat hello2.txt
hello
```

`cat` သည် ဖိုင်များကို ဆက်စပ်ပေးသော ပရိုဂရမ် ဖြစ်သည်။ ထို့ပြင် `>>` ကို အသုံးပြု၍ ဖိုင်၏ အဆုံးတွင် စာသားများ ထပ်ပေါင်းထည့်နိုင်ပါသည်။

ဤ Input/Output redirection ၏ စွမ်းအားကို **Pipes** များတွင် ပိုမို တွေ့မြင်နိုင်ပါသည်။ `|` သင်္ကေတသည် ပရိုဂရမ်တစ်ခု၏ Output ကို အခြား ပရိုဂရမ်တစ်ခု၏ Input အဖြစ် ချိတ်ဆက်ပေးနိုင်ပါသည်။

```
missing:~$ ls -l / | tail -n1
drwxr-xr-x 1 root  root  4096 Jun 20  2019 var
missing:~$ curl --head --silent google.com | grep --ignore-case content-length
| cut --delimiter=' ' -f2
219
```

သုံးရလွယ်ကူပြီး စွမ်းအားထက်မြက်သော tool တစ်ခု (A versatile and powerful tool)

Unix စနစ်များတွင် အထူး ပိုင်ဆိုင်ခွင့်ရှိသော User တစ်ယောက် ရှိပါသည်—ယင်းမှာ **“root”** user ဖြစ်သည်။ Root user သည် စနစ်တစ်ခုလုံးရှိ မည်သည့် ဖိုင်ကိုမဆို ဖန်တီးခြင်း၊ ဖတ်ရှုခြင်း၊ ပြင်ဆင်ခြင်းနှင့် ဖျက်ဆီးခြင်းများ ပြုလုပ်နိုင်သည်။ သို့သော် မတော်တဆ ပျက်စီးမှုများ မဖြစ်စေရန်အတွက် ပုံမှန်အားဖြင့် Root user အဖြစ် Log in မဝင်ဘဲ **`sudo`** command ကို အသုံးပြုကြသည်။ Permission denied error များ ကြုံတွေ့ရပါက Root အဖြစ် လုပ်ဆောင်ရန် လိုအပ်၍ ဖြစ်လေ့ရှိသည်။

ဥပမာအားဖြင့် Laptop စခရင်၏ လင်းထင်းမှုကို `/sys/class/backlight` အောက်ရှိ `brightness` ဖိုင်အတွင်း စာသား ရေးသားခြင်းဖြင့် ပြောင်းလဲနိုင်ပါသည်။

```
$ echo 3 | sudo tee brightness
```

`tee` ပရိုဂရမ်သည် `root` အဖြစ် အလုပ်လုပ်သဖြင့် `/sys` ဖိုင်ကို ရေးသားနိုင်ခြင်း ဖြစ်ပါသည်။

နောက်ထပ် လုပ်ဆောင်ရမည့် အဆင့်များ (Next steps)

ယခုအခါ အခြေခံ အလုပ်များကို လုပ်ဆောင်နိုင်ရန် Shell ကို အသုံးပြုတတ်ပြီ ဖြစ်သည်။ နောက်သင်ခန်းစာတွင် Shell နှင့် အသုံးဝင်သော Command-line ပရိုဂရမ်များကို အသုံးပြု၍ ပိုမို ရှုပ်ထွေးသော အလုပ်များကို မည်သို့ အလိုအလျောက် ခိုင်းစေရမည်ကို သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။

လေ့ကျင့်ခန်းများ (Exercises)

၁။ ဤသင်တန်းအတွက် Bash သို့မဟုတ် ZSH ကဲ့သို့သော Unix Shell တစ်ခု အသုံးပြုရန် လိုအပ်ပါသည်။ Windows အသုံးပြုသူ ဖြစ်ပါက **Windows Subsystem for Linux (WSL)** (https://docs.microsoft.com/en-us/windows/wsl/) သို့မဟုတ် Linux Virtual Machine ကို အသုံးပြုနိုင်သည်။ မိမိ သင့်လျော်သော Shell အသုံးပြုနေကြောင်း စစ်ဆေးရန် `echo $SHELL` ဟု ရိုက်ထည့် စစ်ဆေးပါ။ ၂။ `/tmp` အောက်တွင် `missing` အမည်ရှိ Directory အသစ်တစ်ခု ဖန်တီးပါ။ ၃။ `touch` ပရိုဂရမ်အကြောင်း `man touch` ဖြင့် လေ့လာပါ။ ၄။ `touch` ကို အသုံးပြု၍ `missing` ထဲတွင် `semester` အမည်ရှိ ဖိုင်အသစ်တစ်ခု ဖန်တီးပါ။ ၅။ ထိုဖိုင်အတွင်း အောက်ပါအတိုင်း တစ်ကြောင်းစီ ရေးသားပါ- `#!/bin/sh curl --head --silent https://missing.csail.mit.edu` ၆။ ဖိုင်ကို Execute လုပ်ကြည့်ပါ (ဥပမာ `./semester` ဟု ရိုက်ထည့်ပါ)။ အဘယ်ကြောင့် အလုပ်မလုပ်ကြောင်း `ls` output ကို စစ်ဆေး၍ နားလည်အောင် ကြည့်ပါ။ ၇။ Command ကို Explicit အဖြစ် `sh semester` ဟု Run ကြည့်ပါ။ အဘယ်ကြောင့် `./semester` တုန်းက အလုပ်မလုပ်ဘဲ ယခု အလုပ်လုပ်သနည်း။ ၈။ `chmod` ပရိုဂရမ်အကြောင်း လေ့လာပါ။ (`man chmod`) ၉။ `chmod` ကို အသုံးပြု၍ `sh semester` ရိုက်စရာ မလိုဘဲ `./semester` ဟု တိုက်ရိုက် Run နိုင်အောင် ပြုလုပ်ပါ။ ၁၀။ `|` နှင့် `>` များကို အသုံးပြု၍ `semester` မှ ထွက်ပေါ်လာသော “last modified” ရက်စွဲကို မိမိ Home directory ထဲရှိ `last-modified.txt` ဖိုင်ထဲသို့ ရေးသားပါ။ ၁၁။ `/sys` မှ မိမိ Laptop ဘက်ထရီ ပမာဏ သို့မဟုတ် CPU အပူချိန်ကို ဖတ်ရှုသော Command တစ်ခု ရေးသားပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

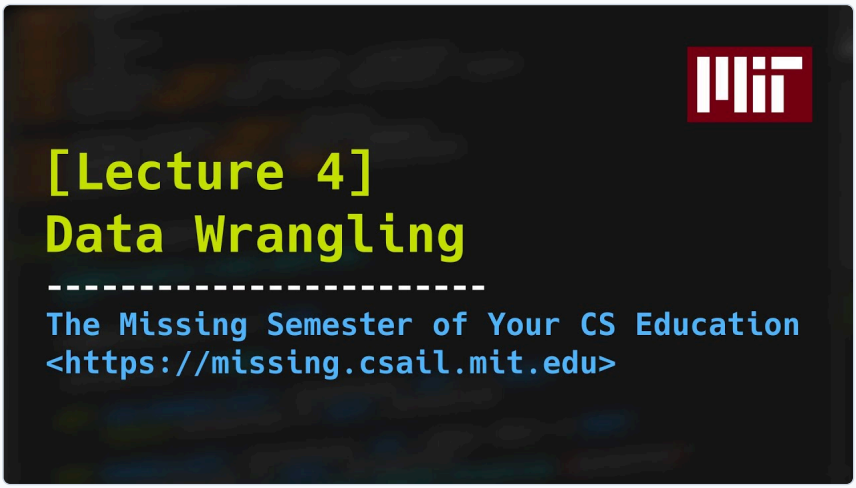
Resource / Description	URL	Micro QR
1. သီးခြား ခေါင်းစဉ်တစ်ခု	https://missing-semester-my.github.io/2020/	
2. Windows Subsystem for Linux (WSL)	https://docs.microsoft.com/en-us/windows/wsl/	

ဒေတာ ပြုပြင်စီမံခြင်း (Data Wrangling)

အနှစ်ချုပ် - sed၊ awk နှင့် Regular Expression များကဲ့သို့သော Command-line tool များကို အသုံးပြု၍ ဒေတာများကို ပြုပြင်ပြောင်းလဲနည်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

ဒေတာ ပြုပြင်စီမံခြင်း (Data Wrangling)



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=sz_dsktIjt4

ဒေတာများကို Format တစ်ခုမှ အခြား Format တစ်ခုသို့ ပြောင်းလဲလိုသည့် အခြေအနေမျိုး ကြုံတွေ့ဖူးပါသလား။ အမှန်တကယ် ကြုံဖူးကြပါလိမ့်မည်။ ဤသင်ခန်းစာသည် ယင်းအကြောင်းကို အဓိကထား သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။ အထူးသဖြင့် စာသား သို့မဟုတ် Binary format ရှိ ဒေတာများကို မိမိ လိုချင်သော ပုံစံအတိုင်း ရရှိသည်အထိ ပြုပြင်ပြင်ဆင်ခြင်း (Data Wrangling) အကြောင်း ဖြစ်ပါသည်။

ယခင် သင်ခန်းစာများတွင် အခြေခံ Data Wrangling အချို့ကို တွေ့မြင်ခဲ့ရပြီး ဖြစ်သည်။ `|` (pipe) operator ကို အသုံးပြုတိုင်း Data Wrangling ကို ပြုလုပ်နေခြင်း ဖြစ်သည်။ ဥပမာ `journalctl | grep -i intel` command ကို ကြည့်ပါ။ ယင်းက စနစ်၏ Log များအနက် Intel ပါဝင်သော စာကြောင်းများကို ရှာဖွေပေးသည်။ ဤသည်မှာ စနစ် log အပြည့်အစုံမှ မိမိအတွက် အသုံးဝင်သော Format သို့ ပြောင်းလဲလိုက်ခြင်း ဖြစ်သည်။ Data Wrangling ၏ အဓိက သဘောတရားမှာ မိမိ ထံတွင် ရှိသော tool များကို မည်သို့ ပေါင်းစပ် အသုံးပြုရမည်ကို တတ်မြောက်ထားခြင်း ဖြစ်သည်။

အစမှ စတင်ကြည့်ကြပါစို့။ Data wrangle ပြုလုပ်ရန်အတွက် အရာနှစ်ခု လိုအပ်သည်—Wrangle ပြုလုပ်မည့် ဒေတာ နှင့် ယင်းဒေတာကို ပြုလုပ်မည့် အရာ တို့ ဖြစ်ကြသည်။ Log ဖိုင်များသည် အလွန်ကောင်းမွန်သော စံနမူနာ ဖြစ်သည်၊ အကြောင်းမှာ ယင်းတို့ကို လေ့လာကြည့်ရှုရန် လိုအပ်သော်လည်း ဖိုင်တစ်ခုလုံးကို ဖတ်ရှုရန် မဖြစ်နိုင်သောကြောင့် ဖြစ်သည်။ အောက်ပါ command ဖြင့် Server ၏ log ကို စစ်ဆေးကြည့်ပါ-

```
ssh myserver journalctl
```

ဤသည်မှာ စာသားများ လွန်စွာ များပြားလှသည်။ SSH နှင့် သက်ဆိုင်သည်များကိုသာ ခွဲထုတ်ကြည့်ကြမည်-

```
ssh myserver journalctl | grep sshd
```

ဒီနေရာမှာ Pipe ကို အသုံးပြု၍ Remote server ပေါ်ရှိ ဖိုင်ကို မိမိ စက်ပေါ်ရှိ `grep` ထံသို့ Stream ပြုလုပ်ပေးပို့ထားခြင်း ဖြစ်သည်။ ဤသည်မှာလည်း အချက်အလက်များ လွန်စွာ များပြားနေပါသေးသည်။ ထို့ကြောင့် အောက်ပါအတိုင်း ပိုမို ကောင်းမွန်အောင် ပြုလုပ်ကြပါစို့-

```
ssh myserver 'journalctl | grep sshd | grep "Disconnected from"' | less
```

အဘယ်ကြောင့် Quote ခံထားသနည်း။ Log များသည် အလွန် ကြီးမားနိုင်သဖြင့် မိမိ စက်ထံသို့ ဒေတာ အားလုံး Stream လုပ်ပြီးမှ Filter လုပ်ခြင်းသည် ကွန်ရက် လှိုင်းနှုန်း အဟောသိကံဟု ဖြစ်စေသည်။ ထို့ကြောင့် Remote server ပေါ်တွင် Filter စတင် ပြုလုပ်ပြီးမှ မိမိ စက်ထံသို့ ပေးပို့ခြင်း ဖြစ်သည်။ `less` သည်

စာသားများကို အထက်အောက် ရွှေ့လျားကြည့်ရှုနိုင်သော Pager ကို ပံ့ပိုးပေးသည်။ ယာယီ ဖိုင်အဖြစ် သိမ်းဆည်း၍လည်း ကြည့်ရှုနိုင်ပါသည်-

```
$ ssh myserver 'journalctl | grep sshd | grep "Disconnected from"' > ssh.log
$ less ssh.log
```

ဤနေရာတွင် မလိုအပ်သော စာသားများ ပါဝင်နေသေးသည်။ ယင်းတို့ကို ဖယ်ရှားရန်အတွက် အလွန်စွမ်းအားထက်မြက်သော tool တစ်ခု ဖြစ်သည့် `sed` ကို လေ့လာကြပါစို့။

`sed` သည် “stream editor” တစ်ခု ဖြစ်သည်။ ယင်းတွင် ဖိုင်ကို တိုက်ရိုက် ပြုပြင်ခြင်းထက် မည်သို့ ပြုပြင်ရမည်ဆိုသော Command တိုများကို ပေးပို့ရသည်။ အသုံးအများဆုံး Command မှာ `s` (substitution - အစားထိုးခြင်း) ဖြစ်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed 's/.*Disconnected from //'
```

ယခု ကျွန်ုပ်တို့ ရေးသားလိုက်သည်မှာ **Regular Expression** (Regex) ဖြစ်သည်။ ယင်းသည် စာသားများကို ပုံစံ (pattern) များနှင့် တိုက်ဆိုင် စစ်ဆေးပေးသော စွမ်းအားထက်မြက်သည့် စနစ်ဖြစ်သည်။ `s` command ၏ ပုံစံမှာ `s/REGEX/SUBSTITUTION/` ဖြစ်သည်။

Regular expressions (ပုံမှန် ဖော်ပြချက်များ)

Regular expression များကို နားလည်ထားခြင်းသည် အလွန် အသုံးဝင်လှပါသည်။ အထက်ပါ ဥပမာ `/.*Disconnected from /` ကို လေ့လာကြည့်ကြပါစို့။

အသုံးများသော Regex Pattern များမှာ- - `.` Newline မှလွဲ၍ မည်သည့် စာလုံးတစ်လုံးမဆို - `*` ရှေ့ စာလုံး 0 ခု သို့မဟုတ် မည်မျှမဆို ပါဝင်ခြင်း - `+` ရှေ့ စာလုံး 1 ခု သို့မဟုတ် မည်မျှမဆို ပါဝင်ခြင်း - `[abc]` `a` ၊ `b` သို့မဟုတ် `c` အနက် စာလုံးတစ်လုံး ပါဝင်ခြင်း - `(RX1|RX2)` `RX1` သို့မဟုတ် `RX2` ကို ကိုက်ညီခြင်း - `^` စာကြောင်း ၏ အစ - `$` စာကြောင်း ၏ အဆုံး

`sed` တွင် အထူး သင်္ကေတများ အဖြစ် အဓိပ္ပာယ်ဖော်ရန် `\` ခံပေးရန် လိုအပ်သည် သို့မဟုတ် `-E` flag ကို အသုံးပြုနိုင်ပါသည်။

Perl command-line တွင် non-greedy matching ပြုလုပ်ရန် `?` ကို အသုံးပြုနိုင်သည်-

```
perl -pe 's/.*?Disconnected from //'
```

ဖိုင် စာကြောင်း တစ်ခုလုံးကို တိုက်ဆိုင် စစ်ဆေးရန်-

```
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user .* [^ ]+ port
[0-9]+( \[preauth\])?$//'
```

မိမိ သိမ်းဆည်းလိုသော စာသားကို Capture Group ဖြင့် သိမ်းဆည်းနိုင်သည်-

```
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
```

Regex များကို စမ်းသပ်ရန် **regex101.com** (https://regex101.com/) ကဲ့သို့သော ဝဘ်ဆိုက်များကို အသုံးပြုနိုင်ပါသည်။

Back to data wrangling (Data wrangling သို့ ပြန်လည် ဝင်ရောက်ခြင်း)

ယခု ကျွန်ုပ်တို့တွင် အောက်ပါ Command ရှိနေပြီ ဖြစ်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
```

မကြာခဏ ရောက်ရှိလာသော အသုံးပြုသူ အမည်များကို စီစဉ် ကြည့်ရှုရန် `sort` နှင့် `uniq -c` ကို အသုံးပြုနိုင်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
 | sort | uniq -c
 | sort -nk1,1 | tail -n10
```

`sort -n` သည် ကိန်းဂဏန်း အစဉ်လိုက် စီစဉ်ပေးပြီး `-k1,1` သည် ပထမဆုံး ကော်လံအတိုင်း စီစဉ်ပေးသည်။

ရလဒ်များကို စာကြောင်း တစ်ကြောင်းစီ မဟုတ်ဘဲ ကော်မာ ခွဲခြားထားသော စာရင်းအဖြစ် ပြောင်းလဲရန် `paste -sd,` ကို အသုံးပြုနိုင်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
 | sort | uniq -c
 | sort -nk1,1 | tail -n10
 | awk '{print $2}' | paste -sd,
```

awk – အခြား Editor တစ်ခု (awk – another editor)

`awk` သည် စာသား သတင်းအချက်အလက် စီးကြောင်းများကို ပြုပြင်ရာတွင် အလွန် ကောင်းမွန်သော ပရိုဂရမ်းမင်း ဘာသာစကား ဖြစ်သည်။

`awk` တွင် `$0` သည် စာကြောင်း တစ်ကြောင်းလုံး ဖြစ်ပြီး `$1` မှ `$n` တို့မှာ ကော်လံ (field) များ ဖြစ်ကြသည်။ `{print $2}` သည် ဒုတိယ ကော်လံကို ထုတ်ပေးခြင်း ဖြစ်သည်။

```
 | awk '$1 == 1 && $2 ~ /^c[^ ]*e$/ { print $2 }' | wc -l
```

`awk` ပရိုဂရမ် တွင် `BEGIN` နှင့် `END` ဘလောက်များကို အသုံးပြု၍ တွက်ချက်မှုများ ပြုလုပ်နိုင်သည်-

```
BEGIN { rows = 0 }
$1 == 1 && $2 ~ /^c[^ ]*e$/ { rows += $1 }
END { print rows }
```

ဒေတာများကို ဆန်းစစ်ခြင်း (Analyzing data)

`bc` ကို အသုံးပြု၍ Shell တွင် သင်္ချာ တွက်ချက်မှုများ ပြုလုပ်နိုင်သည်-

```
  | paste -sd+ | bc -l
```

R (https://www.r-project.org/) ကို အသုံးပြု၍ အချက်အလက် စာရင်းအင်းများကို ဆန်းစစ်နိုင်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
 | sort | uniq -c
 | awk '{print $1}' | R --no-echo -e 'x <- scan(file="stdin", quiet=TRUE); summ
ary(x)'
```

`gnuplot` ကို အသုံးပြု၍ Graph ပုံစံ ရေးဆွဲနိုင်သည်-

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ por
t [0-9]+( \[preauth\])?$/\2/'
 | sort | uniq -c
 | sort -nk1,1 | tail -n10
 | gnuplot -p -e 'set boxwidth 0.5; plot "-" using 1:xtic(2) with boxes'
```

Binary data များကို ပြုပြင်ခြင်း (Wrangling binary data)

`ffmpeg` နှင့် `gzip` များကို အသုံးပြု၍ Binary data များကိုလည်း Pipe ဖြင့် ချိတ်ဆက် ပြုပြင်နိုင်သည်-

```
ffmpeg -loglevel panic -i /dev/video0 -frames 1 -f image2 -
 | convert - -colorspace gray -
 | gzip
 | ssh mymachine 'gzip -d | tee copy.jpg | env DISPLAY=:0 feh -'
```

လေ့ကျင့်ခန်းများ (Exercises)

၁။ **RegexOne** (https://regexone.com/) တွင် Regular Expression လေ့ကျင့်ခန်းများကို လေ့ကျင့်ပါ။ ၂။ `/usr/share/dict/words` ထဲတွင် အနည်းဆုံး `a` ၃ လုံး ပါဝင်ပြီး `'s` ဖြင့် မဆုံးသော စကားလုံး အရေအတွက်ကို ရှာဖွေပါ။ ၃။ `sed s/REGEX/SUBSTITUTION/ input.txt > input.txt` ဟု ရေးသား ခြင်းသည် အဘယ်ကြောင့် မကောင်းသနည်း။ `man sed` တွင် အစားထိုးနည်းကို ရှာဖွေပါ။ ၄။ မိမိ စက်၏ နောက်ဆုံး boot တက်ခဲ့သော စာရင်းများမှ ပျမ်းမျှ၊ မီဒီယံနှင့် အများဆုံး အချိန်ကို ရှာဖွေပါ။ (`journalctl` သို့မဟုတ် `log show` ကို အသုံးပြုပါ)။ ၅။ နောက်ဆုံး ၃ ကြိမ် boot တက်မှုအတွင်း တူညီမှု မရှိသော log စာကြောင်းများကို ရှာဖွေပါ။ ၆။ အွန်လိုင်း ဒေတာ အစုံများမှ `curl` ဖြင့် ဒေတာ ရယူပြီး `jq` သို့မဟုတ် `pup` အသုံးပြု၍ ကော်လံ နှစ်ခုကို ခွဲထုတ်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

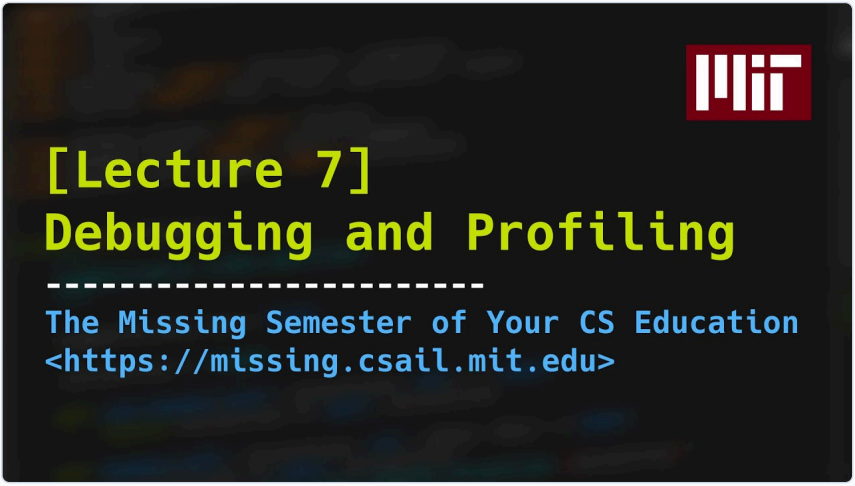
Resource / Description	URL	Micro QR
1. regex101.com	https://regex101.com/	
2. R	https://www.r-project.org/	
3. RegexOne	https://regexone.com/	

Debugging and Profiling

အနှစ်ချုပ် - Logging၊ debuggers နှင့် static analysis တို့ကို အသုံးပြု၍ ပရိဂရမ်များကို Debug ပြုလုပ်ပုံ နှင့် စွမ်းဆောင်ရည်အတွက် Code ကို Profile ပြုလုပ်ပုံများ လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Debugging and 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=l812pUnKxME

ပရိုဂရမ်မင်းတွင် ရွှေရောင် စည်းမျဉ်းတစ်ခု ရှိသည်မှာ - Code သည် သင်မျှော်လင့်သလို အလုပ်လုပ်မည် မဟုတ်ဘဲ သင်ခိုင်းစေသည့်အတိုင်းသာ အလုပ်လုပ်မည် ဖြစ်သည်။ ထို ကွာဟချက်ကို ပေါင်းကူးပေးရန်မှာ အချို့သော အခြေအနေများတွင် အလွန် ခက်ခဲသော စိန်ခေါ်မှု ဖြစ်နိုင်ပါသည်။ ဤသင်ခန်းစာတွင် Bug ပါဝင်နေသော Code များနှင့် Resource စားသုံးမှု များပြားနေသော Code များကို ကိုင်တွယ် ဖြေရှင်းနိုင်မည့် အသုံးဝင်သော နည်းလမ်းနှစ်ခုဖြစ်သည့် Debugging ပြုလုပ်ခြင်း နှင့် Profiling ပြုလုပ်ခြင်း တို့ကို သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။

Debugging

Printf debugging and Logging

“ထိရောက်မှု အရှိဆုံး Debugging tool မှာ သေချာစွာ စဉ်းစားတွေးခေါ်ခြင်း ဖြစ်ပြီး ယင်းနှင့်အတူ သင့်တော်ရာ နေရာများ၌ ထည့်သွင်းထားသော print စာကြောင်းများ ဖြစ်သည်” — Brian Kernighan, *Unix for Beginners*.

ပရိုဂရမ်တစ်ခုကို Debug ပြုလုပ်ရန် ပထမဆုံး နည်းလမ်းမှာ ပြဿနာ တွေ့ရှိရသည့် နေရာများ၌ print စာကြောင်းများ ထည့်သွင်းပြီး ပြဿနာ ဖြစ်ပွားရခြင်း အကြောင်းအရင်းကို နားလည်စေရန် အချက်အလက် လုံလောက်စွာ ရရှိသည်အထိ အကြိမ်ကြိမ် စမ်းသပ်ခြင်း ဖြစ်သည်။

ဒုတိယ နည်းလမ်းမှာ တိုက်ရိုက် print စာကြောင်းများ ရေးသားမည့်အစား သင်၏ ပရိုဂရမ်တွင် Logging ကို အသုံးပြုခြင်း ဖြစ်ပါသည်။ Logging ကို အသုံးပြုခြင်းသည် သာမန် print စာကြောင်းများထက် အကြောင်းကြောင်းကြောင့် ပိုမို ကောင်းမွန်ပါသည် -

- Standard Output သို့ မဟုတ်ဘဲ ဖိုင်များ၊ Socket များ သို့မဟုတ် Remote Server များထံသို့ Log များကို သိမ်းဆည်း ပေးပို့နိုင်ပါသည်။
- Logging သည် သင့်တော်သလို Filter လုပ်နိုင်စေရန် Severity levels များ (ဥပမာ INFO, DEBUG, WARN, ERROR အစရှိသည်) ကို ထောက်ပံ့ပေးပါသည်။
- ပြဿနာ အသစ်များ ဖြစ်ပေါ်သည့်အခါ သင်၏ Log များတွင် မည်သည် မှားယွင်းနေကြောင်း သိရှိနိုင်မည့် အချက်အလက် လုံလောက်စွာ ပါဝင်နိုင်ခြေ ရှိပါသည်။

ဒီနေရာတွင် Log စာတိုများ ရေးသားထားသော နမူနာ Python Code ကို တွေ့နိုင်ပါသည် -

```
$ python logger.py
# Raw output as with just prints
$ python logger.py log
# Log formatted output
$ python logger.py log ERROR
# Print only ERROR levels and above
$ python logger.py color
# Color formatted output
```

Log များကို ပိုမို ဖတ်ရှုရ လွယ်ကူစေရန် ကျွန်ုပ် အနှစ်သက်ဆုံး အကြံပြုချက်တစ်ခုမှာ အရောင်များ (Color code) တပ်ဆင်ခြင်း ဖြစ်ပါသည်။ သင်၏ Terminal တွင် အရာရာကို ဖတ်ရှုရ လွယ်ကူစေရန် အရောင်များ အသုံးပြုထားသည်ကို သတိပြုမိပေမည်။ သို့သော် ၎င်းသည် မည်သို့ အလုပ်လုပ်သနည်း။ `ls` သို့မဟုတ် `grep` ကဲ့သို့သော ပရိုဂရမ်များသည် Terminal အား Output ၏ အရောင်ပြောင်းလဲရန် ညွှန်ကြားသည့် အထူး စာလုံး အစဉ်လိုက်များ ဖြစ်သော **ANSI escape codes** (https://en.wikipedia.org/wiki/ANSI_escape_code) များကို အသုံးပြုကြပါသည်။ ဥပမာအားဖြင့် `echo -e "\e<a href="https://github.com/termstandard/colors#truecolor-support-in-output-devices" class="ext-link">38;2;255;0;0mThis is red\e[0m"` ကို runလိုက်ပါက သင်၏ Terminal က [true color (https://github.com/termstandard/colors#truecolor-support-in-output-devices) ကို ထောက်ပံ့ပါက `This is red` ဟူသော စာတိုကို အနီရောင်ဖြင့် ပြသမည် ဖြစ်သည်။ အကယ်၍ Terminal က ထောက်ခံမှု မရှိပါက (ဥပမာ macOS ၏ Terminal.app) ပိုမို အထွေထွေ ထောက်ပံ့သော အရောင် ၁၆ ရောင် Escape code ကို သုံးနိုင်ပါသည်၊ ဥပမာ `echo -e "\e<a href="https://www.nginx.com/" class="ext-link">31;1mThis is red\e[0m"` ။

အောက်ပါ Script သည် သင်၏ Terminal အတွင်းသို့ RGB အရောင်များစွာ ရိုက်နှိပ်ပြသပုံကို ဖော်ပြထားပါသည် (True color ထောက်ပံ့ပါက)။

```
#!/usr/bin/env bash
for R in $(seq 0 20 255); do
    for G in $(seq 0 20 255); do
        for B in $(seq 0 20 255); do
            printf "\e[38;2;${R};${G};${B}m\e[0m";
        done
    done
done
```

Third party logs

ပိုမို ကြီးမားသော Software စနစ်များကို တည်ဆောက်လာသည့်အခါ သီးခြား ပရိုဂရမ်များအဖြစ် runနေသော Dependencies များနှင့် ကြုံတွေ့ရမည် ဖြစ်ပါသည်။ Web server များ၊ Database များ သို့မဟုတ် Message broker များသည် ဤကဲ့သို့သော Dependency များ၏ အသုံးများသော ဥပမာများ ဖြစ်ကြသည်။ ဤစနစ်များနှင့် မကြာခဏ ချိတ်ဆက် ဆောင်ရွက်သည့်အခါ Client side error စာတိုများ လုံလောက်မှု မရှိနိုင်သဖြင့် ယင်းတို့၏ Log များကို ဖတ်ရှုရန် လိုအပ်တတ်ပါသည်။

ကံကောင်းစွာပင် ပရိုဂရမ် အများစုသည် ယင်းတို့၏ မူပိုင် Log များကို သင်၏ စနစ်အတွင်း တစ်နေရာရာ၌ ရေးသားလေ့ ရှိကြသည်။ UNIX စနစ်များတွင် ပရိုဂရမ်များသည် Log များကို `/var/log` အောက်၌

ရေးသားခြင်းမှာ အလေ့အထ ဖြစ်သည်။ ဥပမာအားဖြင့် [NGINX (https://www.nginx.com/) Webserver သည် ယင်း၏ Log များကို `/var/log/nginx` အောက်တွင် ထားရှိသည်။ နောက်ပိုင်းတွင် စနစ်များသည် **system log** ကို စတင် အသုံးပြုလာကြပြီး သင်၏ Log စာတို အားလုံး ရောက်ရှိရာ နေရာ ဖြစ်လာပါသည်။ Linux စနစ် အများစု (အားလုံးတော့ မဟုတ်ပါ) သည် သင်၏ စနစ်ရှိ Service များ ဖွင့်ထားခြင်း၊ runနေခြင်း အစရှိသည် တို့ကို ထိန်းချုပ်သော System Daemon ဖြစ်သည့် `systemd` ကို အသုံးပြုကြသည်။ `systemd` သည် Log များကို `/var/log/journal` အောက်တွင် သီးသန့် Format ဖြင့် ထားရှိပြီး **`journalctl`** (https://www.man7.org/linux/man-pages/man1/journalctl.1.html) Command ကို အသုံးပြု၍ မက်ဆေ့ဂျ်များကို ပြသနိုင် ပါသည်။ အလားတူ macOS တွင် `/var/log/system.log` ရှိနေဆဲ ဖြစ်သော်လည်း tools အမြောက်အမြား သည် System log ကို သုံးလာကြပြီး **`log show`** (https://www.manpagez.com/man/1/log/) ဖြင့် ကြည့်ရှုနိုင် ပါသည်။ UNIX စနစ် အများစုတွင် Kernel log များကို ကြည့်ရှုရန် **`dmesg`** (https://www.man7.org/linux/man-pages/man1/dmesg.1.html) Command ကိုလည်း အသုံးပြုနိုင်ပါသည်။

System log များအောက်တွင် Logging ပြုလုပ်ရန် **`logger`** (https://www.man7.org/linux/man-pages/man1/logger.1.html) ဟူသော Shell program ကို အသုံးပြုနိုင်ပါသည်။ အောက်တွင် `logger` ကို အသုံးပြုပုံနှင့် System log သို့ ရောက်ရှိသွားခြင်း ရှိမရှိ စစ်ဆေးပုံ နမူနာကို ဖော်ပြထားပါသည်။ ထို့ပြင် ပရို ဂရမ်မင်း ဘာသာစကား အများစုတွင် System log ထို့သို့ ရေးသားရန် Bindings များ ပါရှိကြပါသည်။

```
logger "Hello Logs"
# On macOS
log show --last 1m | grep Hello
# On Linux
journalctl --since "1m ago" | grep Hello
```

Data wrangling သင်ခန်းစာတွင် တွေ့မြင်ခဲ့သည့်အတိုင်း Log များသည် အလွန် ရှည်လျား များပြားနိုင်ပြီး လိုချင်သော အချက်အလက် ရရှိရန် စစ်ထုတ်မှု (filter) ပြုလုပ်ရန် လိုအပ်ပါသည်။ `journalctl` နှင့် `log show` တို့တွင် များစွာ Filter လုပ်နေရပါက ယင်းတို့၏ Output ကို ပထမအဆင့် စစ်ထုတ်ပေးနိုင်သော Flag များကို အသုံးပြုရန် စဉ်းစားနိုင်ပါသည်။ ထို့အပြင် Log ဖိုင်များကို ပိုမို ကောင်းမွန်စွာ ပြသပေးပြီး ကြည့်ရှုနိုင် စေသော **`lnav`** (https://lnav.org/) ကဲ့သို့သော Tool များလည်း ရှိကြပါသည်။

Debuggers

Printf debugging သည် မလုံလောက်တော့သည့်အခါ Debugger တစ်ခုကို အသုံးပြုသင့်ပါသည်။ Debugger ဆိုသည်မှာ ပရိုဂရမ်၏ runနေမှုကို ချိတ်ဆက် ကွပ်ကဲနိုင်သော ပရိုဂရမ် ဖြစ်ပြီး အောက်ပါတို့ကို ပြုလုပ်နိုင် ပါသည် -

- သီးခြား စာကြောင်းသို့ ရောက်ရှိသောအခါ ပရိုဂရမ်၏ အလုပ်လုပ်နေမှုကို ခေတ္တ ရပ်တန့်ခြင်း။

- ပရိုဂရမ်ကို ညွှန်ကြားချက် တစ်ခုချင်းစီအလိုက် တဆင့်ချင်းစီ (step-by-step) သွားရောက်ခြင်း။
- ပရိုဂရမ် ပျက်စီး (crash) သွားပြီးနောက် Variable များ၏ တန်ဖိုးများကို စစ်ဆေးခြင်း။
- သတ်မှတ်ထားသော အခြေအနေ ကိုက်ညီသည့်အခါ ရပ်တန့်ခြင်း။
- နှင့် အခြား အဆင့်မြင့် အင်္ဂါရုပ်များစွာ။

ပရိုဂရမ်မင်း ဘာသာစကား အများစုတွင် Debugger တစ်မျိုးမျိုး ပါဝင်လေ့ ရှိကြပါသည်။ Python တွင် ယင်းသည် Python Debugger `pdb` (https://docs.python.org/3/library/pdb.html) ဖြစ်ပါသည်။

`pdb` က ထောက်ပံ့ပေးသော Command အချို့၏ အတိုချုပ် ရှင်းလင်းချက်မှာ အောက်ပါအတိုင်း ဖြစ်ပါသည် -

- **l**(ist) - လက်ရှိ စာကြောင်း ဘေးပတ်ပတ်လည် ၁၁ ကြောင်းကို ပြသမည်။
- **s**(tep) - လက်ရှိ စာကြောင်းကို runပြီး ပထမဆုံး ရနိုင်သော နေရာတွင် ရပ်မည်။
- **n**(ext) - လက်ရှိ function ၏ နောက် စာကြောင်း ရောက်သည်အထိ သို့မဟုတ် Return ပြန်သည်အထိ ဆက်သွားမည်။
- **b**(reak) - Breakpoint သတ်မှတ်မည်။
- **p**(rint) - လက်ရှိ Context တွင် Expression ကို တွက်ချက်ပြမည်။ `pprint` (https://docs.python.org/3/library/pprint.html) သုံးလိုပါက **pp** လည်း ရှိသည်။
- **r**(eturn) - လက်ရှိ function return ပြန်သည်အထိ ဆက်လက် runမည်။
- **q**(uit) - Debugger မှ ထွက်မည်။

အောက်ပါ Bug ပါဝင်သော Python Code ကို ပြင်ဆင်ရန် `pdb` ကို အသုံးပြုပုံ နမူနာကို လေ့လာကြည့်ပါ။ (သင်ခန်းစာ ဗီဒီယိုကို ကြည့်ပါ)။

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n):
            if arr[j] > arr[j+1]:
                arr[j] = arr[j+1]
                arr[j+1] = arr[j]
    return arr

print(bubble_sort([4, 2, 1, 8, 7, 6]))
```

Python သည် Interpreted language ဖြစ်သောကြောင့် `pdb` shell ကို အသုံးပြု၍ Command များ နှင့် Instruction များကို runနိုင်သည်ကို သတိပြုပါ။ **`ipdb`** (https://pypi.org/project/ipdb/) သည် **`IPython`** (https://ipython.org) REPL ကို အသုံးပြုထားသည့် ပိုမိုကောင်းမွန်သော `pdb` ဖြစ်ပြီး Tab completion, Syntax highlighting, Tracebacks များကို ပေးစွမ်းနိုင်ပါသည်။

Low-level programming အတွက်ဆိုလျှင် **`gdb`** (https://www.gnu.org/software/gdb/) (နှင့် ယင်း၏ **`pwndbg`** (https://github.com/pwndbg/pwndbg)) သို့မဟုတ် **`lldb`** (https://lldb.llvm.org/) တို့ကို ကြည့်ရှုလိုပါလိမ့်မည်။ ယင်းတို့သည် C-like ဘာသာစကား Debugging အတွက် အထူးပြုထားသော်လည်း မည်သည့် Process ကိုမဆို စစ်ဆေးနိုင်ပြီး လက်ရှိ စက်၏ အခြေအနေဖြစ်သော Registers, Stack, Program Counter အစရှိသည်တို့ကို ရယူနိုင်ပါသည်။

Specialized Tools

သင် Debug ပြုလုပ်ရန် ကြိုးစားနေသည်မှာ Black box binary တစ်ခု ဖြစ်နေသည့်တိုင် အကူအညီပေးနိုင်သော Tool များ ရှိပါသည်။ ပရိုဂရမ်များသည် Kernel သာ ပြုလုပ်နိုင်သော လုပ်ဆောင်ချက်များကို ဆောင်ရွက်ရန် လိုအပ်သည့်အခါ **System Calls** (https://en.wikipedia.org/wiki/System_call) များကို အသုံးပြုကြသည်။ သင်၏ ပရိုဂရမ် ပြုလုပ်သော Syscall များကို ခြေရာခံပေးသည့် Command များ ရှိပါသည်။ Linux တွင် **`strace`** (https://www.man7.org/linux/man-pages/man1/strace.1.html) ရှိပြီး macOS နှင့် BSD တွင် **`dtrace`** (https://dtrace.org/about/) ရှိပါသည်။ `dtrace` သည် ယင်း၏ မူပိုင် `D` ဘာသာစကားကို သုံးသဖြင့် သုံးရခက်ခဲနိုင်သော်လည်း `strace` နှင့် ပိုမို တူညီသော Wrapper ဖြစ်သည့် **`dtruss`** (https://www.manpagez.com/man/1/dtruss/) ရှိပါသည် (အသေးစိတ်ကို **ဒီမှာ** (https://8thlight.com/blog/colin-jones/2015/11/06/dtrace-even-better-than-strace-for-osx.html) ကြည့်ပါ)။

အောက်တွင် `ls` ၏ runမှုအတွက် **`stat`** (https://www.man7.org/linux/man-pages/man2/stat.2.html) syscall trace များကို ပြသရန် `strace` သို့မဟုတ် `dtruss` ကို အသုံးပြုပုံ နမူနာကို ဖော်ပြထားပါသည်။ `strace` ကို ပိုမို အသေးစိတ် လေ့လာရန် **ဤဆောင်းပါး** (https://blogs.oracle.com/linux/strace-the-sysadmins-microscope-v2) နှင့် **ဤ zine** (https://jvns.ca/strace-zine-unfolded.pdf) တို့သည် ဖတ်ရှုရန် ကောင်းမွန်ပါသည်။

```
# On Linux
sudo strace -e lstat ls -l > /dev/null
# On macOS
sudo dtruss -t lstat64_extended ls -l > /dev/null
```

အချို့သော အခြေအနေများတွင် သင်၏ ပရိုဂရမ်ရှိ ပြဿနာကို ရှာဖွေရန် Network packets များကို ကြည့်ရှုရန် လိုအပ်နိုင်ပါသည်။ **`tcpdump`** (https://www.man7.org/linux/man-pages/man1/tcpdump.1.html) နှင့် **Wireshark**

(https://www.wireshark.org/) ကဲ့သို့သော Tool များသည် Network packet များ၏ ပါဝင်ပုံကို ဖတ်ရှုနိုင်ပြီး စစ်ထုတ်ပေးနိုင်သော Network packet analyzer များ ဖြစ်ကြပါသည်။

Web development အတွက် Chrome/Firefox ၏ Developer tools များသည် အလွန် အဆင်ပြေပါသည်။ ယင်းတို့တွင် အသုံးဝင်သော Tool များစွာ ပါဝင်ပါသည် - - Source code - မည်သည့် ဝဘ်ဆိုက်၏ HTML/CSS/JS source code ကိုမဆို စစ်ဆေးခြင်း။ - Live HTML, CSS, JS modification - စမ်းသပ်ရန် အတွက် ဝဘ်ဆိုက် အကြောင်းအရာ၊ ပုံစံ၊ အမူအကျင့်များကို ပြောင်းလဲခြင်း။ - Javascript shell - JS REPL တွင် Command များ runခြင်း။ - Network - Request များ၏ အချိန်ဇယားကို ဆန်းစစ်ခြင်း။ - Storage - Cookies နှင့် Local application storage များကို ကြည့်ရှုခြင်း။

Static Analysis

အချို့သော ပြဿနာများအတွက် မည်သည့် Code ကိုမျှ runရန် မလိုပါ။ ဥပမာအားဖြင့် Code ကို သေချာကြည့်ရုံဖြင့် သင်၏ Loop variable သည် ရှိပြီးသား Variable သို့မဟုတ် Function အမည်ကို ဖုံးအုပ် (shadowing) နေသည်ကို သို့မဟုတ် သတ်မှတ်ခြင်း မပြုမီ Variable ကို ဖတ်နေသည်ကို တွေ့ရှိနိုင်ပါသည်။ ဤနေရာတွင် **static analysis** (https://en.wikipedia.org/wiki/Static_program_analysis) tool များ ရောက်ရှိလာပါသည်။ Static analysis ပရိုဂရမ်များသည် Source code ကို လက်ခံပြီး ယင်း၏ မှန်ကန်မှုကို သုံးသပ်ရန် Coding rules များကို သုံး၍ စန်းစစ်ပေးပါသည်။

အောက်ပါ Python snippet တွင် အမှားများစွာ ပါဝင်နေသည်။ ပထမဦးစွာ Loop variable `foo` သည် မူလ function သတ်မှတ်ချက် `foo` ကို Shadowing ပြုလုပ်သွားသည်။ ထို့အပြင် နောက်ဆုံး စာကြောင်းတွင် `bar` အစား `baz` ဟု ရေးသားမိသဖြင့် (၁ မိနစ် ကြာမြင့်မည့်) `sleep` ပြီးနောက် ပရိုဂရမ် Crash ဖြစ်သွားပါမည်။

```
import time

def foo():
    return 42

for foo in range(5):
    print(foo)

bar = 1
bar *= 0.2
time.sleep(60)
print(baz)
```

Static analysis tool များသည် ဤကဲ့သို့သော ပြဿနာများကို ရှာဖွေပေးနိုင်ပါသည်။ `pyflakes` (https://pypi.org/project/pyflakes) ကို runလိုက်ပါက Bug နှစ်ခုစလုံးနှင့် ပတ်သက်သော Error များကို ရရှိမည် ဖြစ်သည်။ `mypy` (https://mypy-lang.org/) သည် Type စစ်ဆေးမှု ပြဿနာများကို ရှာဖွေပေးနိုင်သော အခြား Tool ဖြစ်ပါသည်။ ဤနေရာတွင် `mypy` က `bar` သည် မူလက `int` ဖြစ်ပြီး နောက်ပိုင်း `float` သို့ ပြောင်းလဲသွားကြောင်း သတိပေးမည် ဖြစ်သည်။ ဤပြဿနာ အားလုံးကို Code ကို runစရာ မလိုဘဲ စစ်ဆေးတွေ့ရှိခဲ့ခြင်း ဖြစ်သည်ကို သတိပြုပါ။

```
$ pyflakes foobar.py
foobar.py:6: redefinition of unused 'foo' from line 3
foobar.py:11: undefined name 'baz'

$ mypy foobar.py
foobar.py:6: error: Incompatible types in assignment (expression has type "in
t", variable has type "Callable[[], Any]")
foobar.py:9: error: Incompatible types in assignment (expression has type "floa
t", variable has type "int")
foobar.py:11: error: Name 'baz' is not defined
Found 3 errors in 1 file (checked 1 source file)
```

Shell tools သင်ခန်းစာတွင် Shell script များအတွက် တူညီသော Tool ဖြစ်သည့် `shellcheck` (https://www.shellcheck.net/) အကြောင်းကို သင်ကြားခဲ့ကြပါသည်။

Editor နှင့် IDE အများစုသည် ဤ Tool များ၏ Output ကို Editor အတွင်း၌ တိုက်ရိုက် ပြသပေးခြင်းကို ထောက်ပံ့ပေးပြီး Warning နှင့် Error များကို Highlight ပြပေးပါသည်။ ဤသည်ကို **code linting** ဟု ခေါ်ဆိုကြပြီး စတိုင်လ် ပုံစံ ချိုးဖောက်မှုများ သို့မဟုတ် လုံခြုံရေး မရှိသော ရေးသားမှုများကိုပါ ပြသပေးနိုင်ပါသည်။

Vim တွင် `ale` (https://vimawesome.com/plugin/ale) သို့မဟုတ် `syntastic` (https://vimawesome.com/plugin/syntastic) plugin များက ထိုသို့ ပြုလုပ်ပေးပါသည်။ Python အတွက် `pylint` (https://github.com/PyCQA/pylint) နှင့် `pep8` (https://pypi.org/project/pep8/) တို့သည် စတိုင်လ် Linter များ ဖြစ်ကြပြီး `bandit` (https://pypi.org/project/bandit/) သည် အသုံးများသော လုံခြုံရေး ပြဿနာများကို ရှာဖွေရန် ဒီဇိုင်းထွင်ထားသော Tool ဖြစ်ပါသည်။ အခြား ဘာသာစကားများအတွက် အသုံးဝင်သော Static analysis စာရင်းများကို [Awesome Static Analysis](https://github.com/mre/awesome-static-analysis) တွင် ကြည့်နိုင်ပြီး Linter များအတွက် [Awesome Linters](https://github.com/caramelomartins/awesome-linters) တွင် ကြည့်နိုင်ပါသည်။

Stylistic linting ကို ဖြည့်စွက်ပေးသော Tool များမှာ Python အတွက် `black` (https://github.com/psf/black)၊ Go အတွက် `gofmt` ၊ Rust အတွက် `rustfmt` သို့မဟုတ် JavaScript, HTML, CSS အတွက် `prettier` (https://prettier.io/) ကဲ့သို့သော Code formatters များ ဖြစ်ကြပါသည်။ ဤ Tool များသည် သက်ဆိုင်ရာ

ဘာသာစကား၏ စံနှုန်း စတိုင်လ် ပုံစံများအတိုင်း သင်၏ Code ကို အလိုအလျောက် Format ပြုလုပ်ပေး ပါသည်။

Profiling

သင်၏ Code သည် မျှော်လင့်ထားသည့်အတိုင်း အလုပ်လုပ်သည့်တိုင်အောင် အလုပ်လုပ်နေစဉ်အတွင်း CPU သို့မဟုတ် Memory အားလုံးကို ယူဆောင်သွားပါက မလုံလောက်နိုင်ပါ။ Algorithm အတန်းများတွင် Big O notation အကြောင်းကို သင်ကြားပေးသော်လည်း ပရိုဂရမ်၏ Hot spots များကို မည်သို့ ရှာရမည်ကို မသင်ကြားပေးပါ။ အချိန်မတိုင်မီ အကောင်းဆုံးဖြစ်အောင် ပြုလုပ်ခြင်းသည် ဒုက္ခအားလုံး၏ အမြစ် ဖြစ်သောကြောင့် (https://wiki.c2.com/?PrematureOptimization) Profiler များ နှင့် Monitoring tool များကို လေ့လာသင့်ပါသည်။ ယင်းတို့သည် ပရိုဂရမ်၏ မည်သည့် အစိတ်အပိုင်းက အချိန် သို့မဟုတ် Resource အများဆုံး ယူနေသည်ကို နားလည်အောင် ကူညီပေးမည် ဖြစ်သဖြင့် ထို အစိတ်အပိုင်းများကို အဓိကထား အကောင်းဆုံးဖြစ်အောင် ပြုလုပ်နိုင်ပါမည်။

Timing

Debugging ကဲ့သို့ပင်၊ နေရာနှစ်ခုကြား ကြာမြင့်ချိန်ကို ရိုက်နှိပ်ကြည့်ခြင်းသည် အခြေအနေများစွာတွင် လုံလောက်နိုင်ပါသည်။ အောက်တွင် Python ရှိ `time` (https://docs.python.org/3/library/time.html) module ကို အသုံးပြုထားသော နမူနာ ဖြစ်ပါသည်။

```
import time, random
n = random.randint(1, 10) * 100

# Get current time
start = time.time()

# Do some work
print("Sleeping for {} ms".format(n))
time.sleep(n/1000)

# Compute time between start and now
print(time.time() - start)

# Output
# Sleeping for 500 ms
# 0.5713930130004883
```

သို့သော် နာရီကြည့် ကြာမြင့်ချိန် (Wall clock time) သည် အလှည့်အပတ် ဖြစ်နိုင်သည်၊ အကြောင်းမှာ ကွန်ပျူတာသည် အခြား Process များကို runနေနိုင်သလို Event များကို စောင့်ဆိုင်းနေနိုင်သောကြောင့် ဖြစ်သည်။ Tool များသည် *Real*, *User* နှင့် *Sys* time တို့ကို ခွဲခြားသတ်မှတ်လေ့ ရှိကြပါသည်။ ယေဘုယျ အားဖြင့် *User* + *Sys* သည် ပရိုဂရမ်က CPU တွင် အမှန်တကယ် ကုန်လွန်ခဲ့သော အချိန်ကို ဖော်ပြပေးသည် (အသေးစိတ်ကို ဒီမှာ ကြည့်ပါ (https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1))။

- *Real* - ပရိုဂရမ် စတင်ချိန်မှ ပြီးဆုံးချိန်အထိ နာရီကြည့် ကုန်လွန်ချိန် (အခြား process များ နှင့် စောင့်ဆိုင်းချိန်များ ပါဝင်သည်)
- *User* - User code ကို runရာတွင် CPU ၌ ကုန်လွန်ခဲ့သော အချိန်
- *Sys* - Kernel code ကို runရာတွင် CPU ၌ ကုန်လွန်ခဲ့သော အချိန်

ဥပမာ HTTP request ပြုလုပ်သော Command တစ်ခု၏ ရှေ့တွင် `time` (https://www.man7.org/linux/man-pages/man1/time.1.html) ထည့်၍ runကြည့်ပါ။ လိုင်းနှေးသောအခါ အောက်ပါအတိုင်း Output ရရှိနိုင်ပါသည်။ ဤနေရာတွင် Request ပြီးဆုံးရန် ၂ စက္ကန့်ကျော် ကြာမြင့်ခဲ့သော်လည်း Process သည် CPU user time 15ms နှင့် Kernel CPU time 12ms သာ ကုန်လွန်ခဲ့သည်။

```
$ time curl https://missing.csail.mit.edu &> /dev/null
real    0m2.561s
user    0m0.015s
sys     0m0.012s
```

Profilers

CPU

လူအများစုက *profilers* ဟု ပြောသည့်အခါ အသုံးအများဆုံးဖြစ်သော *CPU profilers* များကို ဆိုလိုခြင်း ဖြစ်သည်။ CPU profiler အဓိက အမျိုးအစား နှစ်ခု ရှိသည် - *tracing* နှင့် *sampling* profilers။ Tracing profiler များသည် ပရိုဂရမ် ပြုလုပ်သော Function call တိုင်းကို မှတ်တမ်းတင်ထားပြီး Sampling profiler များမူ ပရိုဂရမ်ကို ပုံမှန် အချိန်အပိုင်းအခြားအလိုက် (အများအားဖြင့် မီလီစက္ကန့်တိုင်း) စစ်ဆေး၍ Stack ကို မှတ်တမ်းတင်သည်။ ယင်းတို့သည် ပရိုဂရမ်က အချိန်အများဆုံး ကုန်လွန်ခဲ့သည့် စာရင်းဇယားများကို ပြသပေးပါသည်။ ဤဆောင်းပါး (https://jvns.ca/blog/2017/12/17/how-do-ruby—python-profilers-work-) တွင် ပိုမို အသေးစိတ် လေ့လာနိုင်ပါသည်။

ပရိုဂရမ်မင်း ဘာသာစကား အများစုတွင် Code ကို ဆန်းစစ်ရန် Command line profiler ပါဝင်ကြပါသည်။

Python တွင် Function call တစ်ခုစီ၏ အချိန်ကို Profile ပြုလုပ်ရန် `cProfile` module ကို အသုံးပြုနိုင်ပါသည်။ အောက်ပါအတိုင်း Python ဖြင့် ရေးထားသော မူလ Grep နမူနာ ဖြစ်ပါသည် -

```
#!/usr/bin/env python

import sys, re

def grep(pattern, file):
    with open(file, 'r') as f:
        print(file)
        for i, line in enumerate(f.readlines()):
            pattern = re.compile(pattern)
            match = pattern.search(line)
            if match is not None:
                print("{}: {}".format(i, line), end="")

if __name__ == '__main__':
    times = int(sys.argv[1])
    pattern = sys.argv[2]
    for i in range(times):
        for file in sys.argv[3:]:
            grep(pattern, file)
```

အောက်ပါ Command ဖြင့် ဤ Code ကို Profile ပြုလုပ်နိုင်ပါသည်။ Output ကို ဆန်းစစ်ကြည့်ပါက IO က အချိန်အများဆုံး ယူနေပြီး Regex အား Compile လုပ်ခြင်းကလည်း အချိန်အတော်အတန် ယူနေသည်ကို တွေ့ရမည်။ Regex ကို တစ်ကြိမ်သာ Compile လုပ်ရန် လိုသဖြင့် Loop အပြင်သို့ ထုတ်ယူနိုင်ပါသည်။

```
$ python -m cProfile -s tottime grep.py 1000 '^(import|\s*def)[^,]*$' *.py

[omitted program output]

 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     8000    0.266    0.000    0.292    0.000 {built-in method io.open}
     8000    0.153    0.000    0.894    0.000 grep.py:5(grep)
    17000    0.101    0.000    0.101    0.000 {built-in method builtins.print}
     8000    0.100    0.000    0.129    0.000 {method 'readlines' of '_io._IOBa
se' objects}
    93000    0.097    0.000    0.111    0.000 re.py:286(_compile)
    93000    0.069    0.000    0.069    0.000 {method 'search' of '_sre.SRE_Pat
tern' objects}
    93000    0.030    0.000    0.141    0.000 re.py:231(compile)
    17000    0.019    0.000    0.029    0.000 codecs.py:318(decode)
        1    0.017    0.017    0.911    0.911 grep.py:3(<module>)

[omitted lines]
```

Python ၏ `cProfile` ၏ အားနည်းချက်တစ်ခုမှာ Function call တစ်ခုစီအလိုက် အချိန်ကို ပြသခြင်း ဖြစ်သည်။ အထူးသဖြင့် Third-party library များကို သုံးသည့်အခါ ရှုပ်ထွေးသွားနိုင်ပါသည်။ ပိုမို ရှင်းလင်းသော နည်းလမ်းမှာ Code စာကြောင်း တစ်ကြောင်းစီအလိုက် ကုန်လွန်ချိန်ကို ပြသခြင်း ဖြစ်ပြီး *line profilers* များက ထိုသို့ ပြုလုပ်ပေးပါသည်။

ဥပမာ အောက်ပါ Python code သည် အတန်း ဝဘ်ဆိုက်သို့ Request ပို့ပြီး URL များကို စစ်ထုတ်ယူပါသည်
-

```
#!/usr/bin/env python
import requests
from bs4 import BeautifulSoup

# This is a decorator that tells line_profiler
# that we want to analyze this function
@profile
def get_urls():
    response = requests.get('https://missing.csail.mit.edu')
    s = BeautifulSoup(response.content, 'lxml')
    urls = []
    for url in s.find_all('a'):
        urls.append(url['href'])

if __name__ == '__main__':
    get_urls()
```

Python ၏ `cProfile` ကို သုံးပါက Output စာကြောင်း ၂၅၀၀ ကျော် ရရှိမည် ဖြစ်သည်။ **`line_profiler`** (https://github.com/pyutils/line_profiler) ကို runလိုက်ပါက စာကြောင်းတစ်ကြောင်းစီ၏ အချိန်ကို တွေ့ရမည် -

```
$ kernprof -l -v a.py
Wrote profile results to urls.py.lprof
Timer unit: 1e-06 s

Total time: 0.636188 s
File: a.py
Function: get_urls at line 5

Line #  Hits         Time  Per Hit   % Time  Line Contents
==============================================================
 5                                           @profile
 6                                           def get_urls():
 7         1     613909.0 613909.0     96.5      response = requests.get('http
s://missing.csail.mit.edu')
 8         1      21559.0  21559.0      3.4      s = BeautifulSoup(response.con
tent, 'lxml')
 9         1          2.0      2.0      0.0      urls = []
10        25        685.0     27.4      0.1      for url in s.find_all('a'):
11        24         33.0      1.4      0.0          urls.append(url['href'])
```

Memory

C သို့မဟုတ် C++ ကဲ့သို့သော ဘာသာစကားများတွင် Memory leaks များသည် မလိုအပ်တော့သော Memory များကို ပယ်ဖျက်မပေးဘဲ ဖြစ်စေနိုင်ပါသည်။ Memory debugging အတွက် **Valgrind** (https://valgrind.org/) ကဲ့သို့သော Tool များကို အသုံးပြု၍ Memory leaks များကို ရှာဖွေနိုင်ပါသည်။

Python ကဲ့သို့ Garbage collected ဘာသာစကားများတွင်လည်း Memory profiler ကို သုံးရန် အသုံးဝင်ပါသည်၊ အကြောင်းမှာ Memory ထဲရှိ Object များသို့ Pointer များ ရှိနေသရွေ့ Garbage collection ပြုလုပ်မည် မဟုတ်သောကြောင့် ဖြစ်သည်။ အောက်တွင် **memory-profiler** (https://pypi.org/project/memory-profiler/) ဖြင့် runထားသော နမူနာ ဖြစ်ပါသည် -

```
@profile
def my_func():
    a = [1] * (10 ** 6)
    b = [2] * (2 * 10 ** 7)
    del b
    return a

if __name__ == '__main__':
    my_func()
```

```
$ python -m memory_profiler example.py
Line #    Mem usage    Increment   Line Contents
================================================
     3                             @profile
     4      5.97 MB    0.00 MB   def my_func():
     5     13.61 MB    7.64 MB       a = [1] * (10 ** 6)
     6    166.20 MB  152.59 MB       b = [2] * (2 * 10 ** 7)
     7     13.61 MB -152.59 MB       del b
     8     13.61 MB    0.00 MB       return a
```

Event Profiling

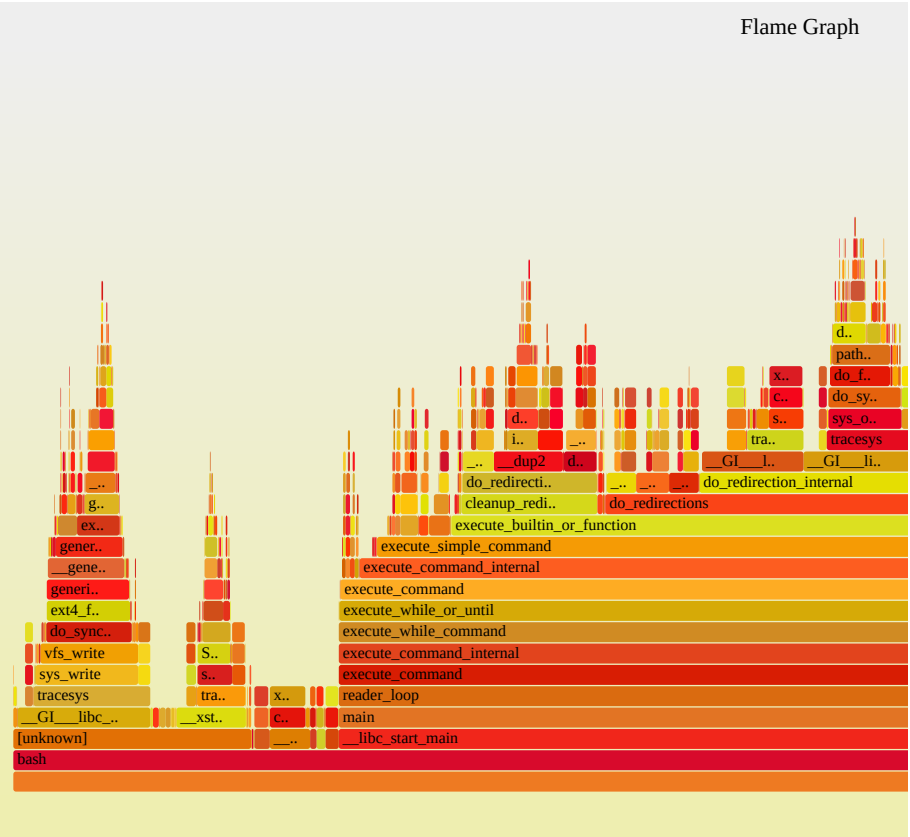
Debugging အတွက် `strace` ကဲ့သို့ပင်၊ Profile ပြုလုပ်သည့်အခါ Code ၏ အသေးစိတ်ကို လျစ်လျူရှုပြီး Black box အဖြစ် သတ်မှတ်လိုနိုင်ပါသည်။ **`perf`** (https://www.man7.org/linux/man-pages/man1/perf.1.html) Command သည် CPU ကွဲပြားမှုများကို ကျော်လွှန်၍ Time သို့မဟုတ် Memory မဟုတ်ဘဲ System events များကို အစီရင်ခံပေးပါသည်။ ဥပမာ `perf` သည် နည်းပါးသော Cache locality၊ များပြားသော Page faults များကို အလွယ်တကူ ပြသပေးနိုင်ပါသည်။

- `perf list` - perf ဖြင့် ခြေရာခံနိုင်သော Event စာရင်းများကို ပြသည်
- `perf stat COMMAND ARG1 ARG2` - Process သို့မဟုတ် Command နှင့် သက်ဆိုင်သော Event ရေတွက်မှုများကို ရယူသည်
- `perf record COMMAND ARG1 ARG2` - Command ၏ runမှုကို မှတ်တမ်းတင်၍ `perf.data` ဖိုင်ထဲ သိမ်းဆည်းသည်
- `perf report` - `perf.data` ရှိ အချက်အလက်များကို ပြသပေးသည်

Visualization

စက္ကူပေါ်မှ Profiler output များသည် အလွန် များပြားနိုင်ပါသည်။ လူများသည် ပုံရိပ်ကြည့် သတ္တဝါများ ဖြစ်ကြပြီး ကိန်းဂဏန်း အများအပြားကို ဖတ်ရသည်ကို သဘောမကျကြပါ။ ထို့ကြောင့် Profiler output များကို လွယ်ကူစွာ ကြည့်ရှုနိုင်သည့် Tool များ ရှိကြပါသည်။

အသုံးများသော နည်းလမ်းတစ်ခုမှာ Y axis တွင် Function call အစဉ်လိုက်နှင့် X axis တွင် ကြာမြင့်ချိန်ကို ပြသပေးသော **Flame Graph** (https://www.brendangregg.com/flamegraphs.html) ကို အသုံးပြုခြင်း ဖြစ်ပါသည်။



[Call graph](#) သို့မဟုတ် Control flow graph များသည် Function များကို Node များအဖြစ်နှင့် Call များကို Directed Edge များအဖြစ် ပြသပေးပါသည်။ Python တွင် `pycallgraph` (https://pycallgraph.readthedocs.io/) library ဖြင့် ယင်းတို့ကို ထုတ်ယူနိုင်ပါသည်။

![Call Graph](https://upload.wikimedia.org/wikipedia/commons/2/2f/A_Call_Graph_generated_by_pycallgraph.png)

Resource Monitoring

တစ်ခါတစ်ရံ ပရိုဂရမ်၏ စွမ်းဆောင်ရည်ကို ဆန်းစစ်ရန် ပထမအဆင့်မှာ ယင်း၏ အမှန်တကယ် Resource စားသုံးမှုကို နားလည်ခြင်း ဖြစ်သည်။ CPU usage, Memory usage, Network, Disk usage များကို ကြည့်ရှုရန် Command line tool များစွာ ရှိပါသည် -

- **General Monitoring** - အသုံးအများဆုံးမှာ `top` (https://www.man7.org/linux/man-pages/man1/top.1.html) ၏ ပိုမိုကောင်းမွန်သော မူကွဲဖြစ်သည့် `htop` (https://htop.dev/) ဖြစ်သည်။ `<F6>` ဖြင့် Process များ စီနိုင်သည်၊ `t` ဖြင့် Tree ပြနိုင်သည်၊ `h` ဖြင့် Thread များ ပိတ်/ဖွင့် နိုင်သည် ။ UI ကောင်းမွန်သော `glances` (https://nicolargo.github.io/glances/) ကိုလည်း ကြည့်ပါ။ Real-time resource metric များအတွက် `dool` (https://github.com/scottchiefbaker/dool) လည်း ရှိသည်။
- **I/O operations** - `iotop` (https://www.man7.org/linux/man-pages/man8/iotop.8.html) သည် လက်ရှိ I/O အသုံးပြုမှုကို ပြသသည်။
- **Disk Usage** - `df` (https://www.man7.org/linux/man-pages/man1/df.1.html) သည် Partition အလိုက် ပြသပြီး `du` (https://man7.org/linux/man-pages/man1/du.1.html) သည် ဖိုင်အလိုက် **d**isk **u**sage ကို ပြသည်။ `-h` flag ဖြင့် Human readable format ရနိုင်သည်။ ၎င်းထက် ပိုမို အဆင်ပြေသော Interactive မုဒ်မှာ `ncdu` (https://dev.yorhel.nl/ncdu) ဖြစ်သည်။
- **Memory Usage** - `free` (https://www.man7.org/linux/man-pages/man1/free.1.html) သည် စနစ်၏ မလွတ်သေးသော နှင့် လွတ်နေသော Memory ကို ပြသည်။
- **Open Files** - `lsof` (https://www.man7.org/linux/man-pages/man8/lsof.8.html) သည် Process များ ဖွင့်ထားသော ဖိုင် သတင်းအချက်အလက်များကို စာရင်းပြသည်။
- **Network Connections and Config** - `ss` (https://www.man7.org/linux/man-pages/man8/ss.8.html) သည် Network packet စာရင်းဇယားများကို စောင့်ကြည့်နိုင်ပြီး သီးခြား Port ကို မည်သည့် Process သုံးနေသည်ကို ရှာနိုင်သည်။ Routing နှင့် Interface များအတွက် `ip` (https://man7.org/linux/man-pages/man8/ip.8.html) ကို သုံးနိုင်သည်။
- **Network Usage** - `nethogs` (https://github.com/raboof/nethogs) နှင့် `iftop` (https://pdw.ex-parrot.com/iftop/) တို့သည် Network အသုံးပြုမှုကို စောင့်ကြည့်ရန် ကောင်းမွန်သော Interactive CLI tool များ ဖြစ်သည်။

စမ်းသပ်ရန်အတွက် `stress` (https://linux.die.net/man/1/stress) command ဖြင့် Load များကို ဖန်တီးနိုင်ပါသည်။

Specialized tools

Black box benchmarking ပြုလုပ်ရန် `hyperfine` (https://github.com/sharkdp/hyperfine) ကဲ့သို့သော Tool များကို သုံး၍ Command line ပရိုဂရမ်များကို အမြန် စမ်းသပ်နိုင်ပါသည်။

```
$ 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"'
```

Browser များတွင်လည်း စွမ်းဆောင်ရည် စမ်းသပ်ရန် Tool များစွာ ပါဝင်ပါသည် (**Firefox** (https://profiler.firefox.com/docs/) နှင့် **Chrome** (https://developers.google.com/web/tools/chrome-devtools/rendering-tools))။

Exercises

Debugging

1. Linux တွင် `journalctl` သို့မဟုတ် macOS တွင် `log show` ကို သုံး၍ လွန်ခဲ့သော နေ့ရက်အတွင်း Superuser ဝင်ရောက်မှုများကို ကြည့်ပါ။ မရှိပါက `sudo ls` runပြီး ပြန်စစ်ပါ။

2. `pdb` သင်ခန်းစာ (https://github.com/spiside/pdb-tutorial) ကို ပြုလုပ်ပါ။ ဤဆောင်းပါး (https://realpython.com/python-debugging-pdb) ကိုလည်း ဖတ်ပါ။

3. `shellcheck` (https://www.shellcheck.net/) ကို တပ်ဆင်ပြီး အောက်ပါ Script ကို စစ်ဆေးပါ။ ဘာမှားနေသနည်း ပြင်ပါ။

```
#!/bin/sh
## Example: a typical script with several problems
for f in $(ls *.m3u)
do
  grep -qi hq.*mp3 $f \
    && echo -e 'Playlist $f contains a HQ file in mp3 format'
done
```

4. (အဆင့်မြင့်) Reversible debugging (https://undo.io/resources/reverse-debugging-whitepaper/) အကြောင်း ဖတ်ပြီး `rr` (https://rr-project.org/) သို့မဟုတ် `RevPDB` (https://morepypy.blogspot.com/2016/07/reverse-debugging-for-python.html) ကို သုံးပါ။

Profiling

1. ဒီမှာ Sorting algorithm များ ရှိပါသည်။ `cProfile` (https://docs.python.org/3/library/profile.html) နှင့် `line_profiler` (https://github.com/pyutils/line_profiler) တို့ကို သုံး၍ Insertion sort နှင့် Quicksort ကြာမြင့်ချိန် ယှဉ်ပါ။ `memory_profiler` ကို သုံး၍ Memory စားသုံးမှုကို စစ်ပါ။

2. အောက်ပါ Python code ဖြင့် Fibonacci ကိန်းများ တွက်ချက်ပုံကို လေ့လာပါ -

```
#!/usr/bin/env python
def fib0(): return 0

def fib1(): return 1

s = """def fib{}(): return fib{}() + fib{}()"""

if __name__ == '__main__':

    for n in range(2, 10):
        exec(s.format(n, n-1, n-2))
    # from functools import lru_cache
    # for n in range(10):
    #     exec("fib{} = lru_cache(1)(fib{})".format(n, n))
    print(eval("fib9()"))
```

`pycallgraph graphviz -- ./fib.py` ဖြင့် run၍ `pycallgraph.png` ကို စစ်ပါ။ `fib0` ကို ဘယ်နှစ်ကြိမ် ခေါ်သနည်း။ Comment ဖြုတ်၍ ပြန်runပါ၊ အခု ဘယ်နှစ်ကြိမ် ခေါ်သနည်း။

3. Port 4444 တွင် `python -m http.server 4444` စတင် runပါ။ အခြား Terminal မှ `lsof | grep LISTEN` ဖြင့် စစ်ပါ၊ PID ကို ရှာပြီး `kill <PID>` ဖြင့် ပိတ်ပါ။

4. `stress -c 3` runပြီး `htop` ဖြင့် ကြည့်ပါ။ `taskset --cpu-list 0,2 stress -c 3` runပြီး ကြည့်ပါ။ `man taskset` ကို ဖတ်ပါ။ စိန်ခေါ်မှု- **`cgroups`** (https://www.man7.org/linux/man-pages/man7/cgroups.7.html) ကို သုံး၍ ပြုလုပ်ပါ။ `stress -m` ဖြင့် memory စားသုံးမှုကို ကန့်သတ် ကြည့်ပါ။

5. (အဆင့်မြင့်) `curl ipinfo.io` runပြီး Wireshark ဖြင့် Packet များကို ဖမ်းယူကြည့်ပါ (`http` filter သုံးပါ)။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

```
</tbody>
```

Resource / Description	URL	Micro QR
1. ANSI escape codes	https://en.wikipedia.org/wiki/ANSI_escape_code	
2. 38;2;255;0;0mThis is red\e[0m"` ကို runလိုက်ပါက သင်၏ Terminal က [true color	https://github.com/termstandard/colors#truecolor-support-in-output-devices	
3. 31;1mThis is red\e[0m"`။ အောက်ပါ Script သည် သင်၏ Terminal အတွင်းသို့ RGB အရောင်များစွာ ရိုက်နှိပ်ပြသပုံကို ဖော်ပြထားပါသည် (True color ထောက်ပံ့ပါက)။ ```bash #!/usr/bin/env bash for R in $(seq 0 20 255); do for G in $(seq 0 20 255); do for B in $(seq 0 20 255); do printf "\e[38;2;${R};${G};${B}m\e[0m"; done done done ## Third party logs ပိုမို ကြီးမားသော Software စနစ်များကို တည်ဆောက်လာသည့်အခါ သီးခြား ပရိုဂရမ်များအဖြစ် runနေသော Dependencies များနှင့် ကြုံတွေ့ရမည် ဖြစ်ပါသည်။ Web server များ၊ Database များ သို့မဟုတ် Message broker များသည် ဤကဲ့သို့သော Dependency များ၏ အသုံးများသော ဥပမာများ ဖြစ်ကြသည်။ ဤစနစ်များနှင့် မကြာခဏ ချိတ်ဆက်ဆောင်ရွက်သည့်အခါ Client side error စာတိုများ လုံလောက်မှု မရှိနိုင်သဖြင့် ယင်းတို့၏ Log များကို ဖတ်ရှုရန် လိုအပ်တတ်ပါသည်။ ကံကောင်းစွာပင် ပရိုဂရမ် အများစုသည် ယင်းတို့၏ မူပိုင် Log များကို သင်၏ စနစ်အတွင်း တစ်နေရာရာ၌ ရေးသားလေ့ ရှိကြသည်။ UNIX စနစ်များတွင် ပရိုဂရမ်များသည် Log များကို `/var/log` အောက်၌ ရေးသားခြင်းမှာ အလေ့အထ ဖြစ်သည်။ ဥပမာ အားဖြင့် [NGINX	https://www.nginx.com/	

Resource / Description	URL	Micro QR
4. `journalctl`	https://www.man7.org/linux/man-pages/man1/journalctl.1.html	
5. `log show`	https://www.manpagez.com/man/1/log/	
6. `dmesg`	https://www.man7.org/linux/man-pages/man1/dmesg.1.html	
7. `logger`	https://www.man7.org/linux/man-pages/man1/logger.1.html	
8. `lnav`	https://lnav.org/	
9. `pdb`	https://docs.python.org/3/library/pdb.html	
10. `pprint`	https://docs.python.org/3/library/pprint.html	
11. `ipdb`	https://pypi.org/project/ipdb/	
12. `IPython`	https://ipython.org	
13. `gdb`	https://www.gnu.org/software/gdb/	
14. `pwndbg`	https://github.com/pwndbg/pwndbg	
15. `lldb`	https://lldb.llvm.org/	
16. System Calls	https://en.wikipedia.org/wiki/System_call	
17. `strace`	https://www.man7.org/linux/man-pages/man1/strace.1.html	
18. `dtrace`	https://dtrace.org/about/	
19. `dtruss`	https://www.manpagez.com/man/1/dtruss/	

Resource / Description	URL	Micro QR
20. ဒီမှာ	https://8thlight.com/blog/colin-jones/2015/11/06/dtrace-even-better-than-strace-for-osx.html	
21. `stat`	https://www.man7.org/linux/man-pages/man2/stat.2.html	
22. ဤဆောင်းပါး	https://blogs.oracle.com/linux/strace-the-sysadmins-microscope-v2	
23. ဤ zine	https://jvns.ca/strace-zine-unfolded.pdf	
24. `tcpdump`	https://www.man7.org/linux/man-pages/man1/tcpdump.1.html	
25. Wireshark	https://www.wireshark.org/	
26. static analysis	https://en.wikipedia.org/wiki/Static_program_analysis	
27. `pyflakes`	https://pypi.org/project/pyflakes	
28. `mypy`	https://mypy-lang.org/	
29. `shellcheck`	https://www.shellcheck.net/	
30. `ale`	https://vimawesome.com/plugin/ale	
31. `syntastic`	https://vimawesome.com/plugin/syntastic	
32. `pylint`	https://github.com/PyCQA/pylint	
33. `pep8`	https://pypi.org/project/pep8/	
34. `bandit`	https://pypi.org/project/bandit/	
35. Awesome Static Analysis	https://github.com/mre/awesome-static-analysis	

Resource / Description	URL	Micro QR
36. Awesome Linters	https://github.com/caramelomartins/awesome-linters	
37. `black`	https://github.com/psf/black	
38. `prettier`	https://prettier.io/	
39. အချိန်မတိုင်မီ အကောင်းဆုံးဖြစ်အောင် ပြုလုပ်ခြင်းသည် ဒုက္ခအားလုံး၏ အမြစ် ဖြစ်သောကြောင့်	https://wiki.c2.com/?PrematureOptimization	
40. `time`	https://docs.python.org/3/library/time.html	
41. အသေးစိတ်ကို ဒီမှာ ကြည့်ပါ	https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1	
42. `time`	https://www.man7.org/linux/man-pages/man1/time.1.html	
43. ဤဆောင်းပါး	https://jvns.ca/blog/2017/12/17/how-do-ruby—python-profilers-work-	
44. `line_profiler`	https://github.com/pyutils/line_profiler	
45. Valgrind	https://valgrind.org/	
46. memory-profiler	https://pypi.org/project/memory-profiler/	
47. `perf`	https://www.man7.org/linux/man-pages/man1/perf.1.html	
48. Flame Graph	https://www.brendangregg.com/flamegraphs.html	
49. ![FlameGraph	https://www.brendangregg.com/FlameGraphs/cpu-bash-flamegraph.svg	
50. `pycallgraph`	https://pycallgraph.readthedocs.io/	
51. Call Graph	https://upload.wikimedia.org/wikipedia/commons/2/2f/A_Call_Graph_generated_by_pycallgraph.png	

Resource / Description	URL	Micro QR
52. `top`	https://www.man7.org/linux/man-pages/man1/top.1.html	
53. `htop`	https://htop.dev/	
54. `glances`	https://nicolargo.github.io/glances/	
55. `dool`	https://github.com/scottchiefbaker/dool	
56. `iotop`	https://www.man7.org/linux/man-pages/man8/iotop.8.html	
57. `df`	https://www.man7.org/linux/man-pages/man1/df.1.html	
58. `du`	https://man7.org/linux/man-pages/man1/du.1.html	
59. `ncdu`	https://dev.yorhel.nl/ncdu	
60. `free`	https://www.man7.org/linux/man-pages/man1/free.1.html	
61. `lsof`	https://www.man7.org/linux/man-pages/man8/lsof.8.html	
62. `ss`	https://www.man7.org/linux/man-pages/man8/ss.8.html	
63. `ip`	https://man7.org/linux/man-pages/man8/ip.8.html	
64. `nethogs`	https://github.com/raboof/nethogs	
65. `iftop`	https://pdw.ex-parrot.com/iftop/	
66. `stress`	https://linux.die.net/man/1/stress	
67. `hyperfine`	https://github.com/sharkdp/hyperfine	

Resource / Description	URL	Micro QR
68. Firefox	https://profiler.firefox.com/docs/	
69. Chrome	https://developers.google.com/web/tools/chrome-devtools/rendering-tools	
70. သင်ခန်းစာ	https://github.com/spiside/pdb-tutorial	
71. ဤဆောင်းပါး	https://realpython.com/python-debugging-pdb	
72. Reversible debugging	https://undo.io/resources/reverse-debugging-whitepaper/	
73. `rr`	https://rr-project.org/	
74. `RevPDB`	https://morepypy.blogspot.com/2016/07/reverse-debugging-for-python.html	
75. `cProfile`	https://docs.python.org/3/library/profile.html	
76. `cgroups`	https://www.man7.org/linux/man-pages/man7/cgroups.7.html	

```
\pagebreak

# Editors (Vim)

> **အနှစ်ချုပ်** - ထိရောက်သော ကုဒ် ပြင်ဆင်ရေးသားမှုအတွက် ရည်ရွယ်ထုတ်လုပ်ထားသည့် စွမ်းအားထက်မြက်သော Text Editor ဖြစ်သည့် Vim ကို အသုံးပြုနည်း လေ့လာပါ။


```{=html}
<div class="video-card">
  <div class="video-card-header">
    <span class="video-badge">▶ LECTURE VIDEO REFERENCE</span>
    <h3 class="video-title">Editors (Vim)</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_a6Q8Na575qc.jpg" alt="Editors (Vim) 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/missing-semester-my.github.io/ebook/assets/qrcodes/yt_a6Q8Na575qc.png" alt="QR Code for Editors (Vim)" 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=a6Q8Na575qc" class="video-url">https://www.youtube.com/watch?v=a6Q8Na575qc</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

အင်္ဂလိပ် စာသားများ ရေးသားခြင်းနှင့် ကုဒ် (Code) ရေးသားခြင်းတို့သည် အလွန် ကွဲပြားသော လုပ်ဆောင်ချက်များ ဖြစ်ကြသည်။ ပရိုဂရမ်းမင်း ရေးသားသည့်အခါ စာသားများ အများအပြား ဆက်တိုက် ရေးသားခြင်းထက် ဖိုင်များအကြား ကူးပြောင်းခြင်း၊ စာဖတ်ခြင်း၊ လမ်းကြောင်းရှာခြင်းနှင့် ကုဒ်များကို ပြင်ဆင်ခြင်းတို့တွင် အချိန်ပိုမို ကုန်လွန်လေ့ရှိသည်။ ထို့ကြောင့် အင်္ဂလိပ်စာ ရေးသားရန် ပရိုဂရမ်များနှင့် ကုဒ်
```

ရေးသားရန် ပရိုဂရမ်များ ကွဲပြားနေခြင်းမှာ အလွန် သဘာဝကျပါသည် (ဥပမာ Microsoft Word နှင့် Visual Studio Code)။

Programmer များအနေဖြင့် ကျွန်ုပ်တို့သည် အချိန်အများစုကို ကုဒ် ပြင်ဆင်ရေးသားရာတွင် ကုန်လွန်ကြရသဖြင့်၊ မိမိ လိုအပ်ချက်နှင့် ကိုက်ညီသော Editor တစ်ခုကို ကျွမ်းကျင်အောင် လေ့လာခြင်းသည် အလွန် တန်ဖိုးရှိပါသည်။ Editor အသစ်တစ်ခုကို လေ့လာနည်းမှာ အောက်ပါအတိုင်း ဖြစ်သည်-

- သင်ခန်းစာမှ စတင်ပါ (ဤ သင်ခန်းစာ နှင့် ကျွန်ုပ်တို့ ညွှန်းဆိုထားသော အရင်းအမြစ်များ)
- စတင်ချိန်တွင် ပိုမို နှေးကွေးနိုင်သော်လည်း မိမိ စာသား ပြင်ဆင်မှု အားလုံးအတွက် ထို Editor ကိုပင် ဆက်လက် အသုံးပြုပါ
- လေ့လာရင်း ရှာဖွေပါ: ပိုမို ကောင်းမွန်သော နည်းလမ်း ရှိသင့်သည်ဟု ထင်မြင်ပါက အမှန်တကယ် ပိုမို ကောင်းမွန်သော နည်းလမ်း ရှိနေတတ်ပါသည်။

Editor အသစ်တစ်ခုကို အသုံးပြုရာတွင် ပထမ ၁ နာရီ သို့မဟုတ် ၂ နာရီအတွင်း ဖိုင် ဖွင့်ခြင်း၊ ပြင်ဆင်ခြင်း၊ သိမ်းဆည်းခြင်း/ထွက်ခြင်း စသည့် အခြေခံများကို တတ်မြောက်မည် ဖြစ်သည်။ နာရီ ၂၀ ခန့် အသုံးပြုပြီးပါက ယခင် Editor ကဲ့သို့ပင် မြန်ဆန်လာမည် ဖြစ်ပြီး၊ ထိုနောက်ပိုင်းတွင် ယင်း၏ အကျိုးကျေးဇူးများကို စတင် ခံစားရမည် ဖြစ်သည်—Editor အသစ်ကို အသုံးပြုခြင်းဖြင့် အချိန်များစွာ အသက်သာစေမည် ဖြစ်သည်။

မည်သည့် Editor ကို လေ့လာမလဲ? (Which editor to learn?)

ယနေ့ခေတ် လူကြိုက်များသော Editor များအနက် **Visual Studio Code** (https://code.visualstudio.com/) သည် လူကြိုက်အများဆုံး Editor ဖြစ်ပြီး၊ **Vim** (https://www.vim.org/) သည် Command-line အခြေပြု လူကြိုက်အများဆုံး Editor ဖြစ်သည်။

Vim

ဤသင်တန်း၏ သင်ကြားသူ အားလုံးသည် Vim ကို အသုံးပြုကြသည်။ Vim သည် Vi editor (1976) မှ စတင်ခဲ့ပြီး ယနေ့ထက်တိုင် တိုးတက်လျက် ရှိသည်။ Vim တွင် အလွန် ကောင်းမွန်သော အတွေးအခေါ်များ ပါဝင်သဖြင့် Tool အများအပြားက Vim Mode ကို ပံ့ပိုးပေးထားကြသည်။ Vim ၏ လုပ်ဆောင်ချက် အားလုံးကို မိနစ် ၅၀ အတွင်း သင်ကြားပေးရန် မဖြစ်နိုင်သော်လည်း၊ Vim ၏ အတွေးအခေါ်နှင့် အခြေခံများကို ရှင်းလင်း သင်ကြားပေးသွားမည် ဖြစ်ပါသည်။

Vim ၏ အတွေးအခေါ် (Philosophy of Vim)

ပရိုဂရမ်းမင်း ရေးသားရာတွင် အချိန်အများစုကို စာသားအသစ် ရေးသားခြင်းထက် ဖတ်ရှုခြင်းနှင့် ပြင်ဆင်ခြင်းတို့တွင် ကုန်လွန်ကြသည်။ ထို့ကြောင့် Vim သည် **Modal editor** ဖြစ်သည်—စာသား ထည့်သွင်းခြင်းနှင့် စာသား ပြင်ဆင်ခြင်းတို့အတွက် ကွဲပြားသော Mode များကို အသုံးပြုထားသည်။ Vim ၏ Interface ကိုယ်တိုင်သည် ပရိုဂရမ်းမင်း ဘာသာစကားတစ်ခုကဲ့သို့ အလုပ်လုပ်သည်—ခလုတ်များ နှိပ်ခြင်း (Keystrokes) သည် Command များ ဖြစ်ကြပြီး ယင်း Command များကို ပေါင်းစပ် အသုံးပြုနိုင်ပါသည်။ Vim သည် Mouse နှင့် Arrow Key များ အသုံးပြုခြင်းကို ရှောင်ရှားထားသဖြင့် မိမိ တွေးခေါ်သည့် အရှိန်အတိုင်း ရေးသား ပြင်ဆင်နိုင်စေမည် ဖြစ်သည်။

Modal editing (Mode မျိုးစုံဖြင့် ပြင်ဆင်ခြင်း)

Vim တွင် လုပ်ဆောင်ချက် Mode မျိုးစုံ ပါရှိပါသည်-

- **Normal**: ဖိုင်အတွင်း သွားလာရန်နှင့် ပြင်ဆင်ရန်
- **Insert**: စာသားများ ထည့်သွင်းရန်
- **Replace**: စာသားများကို အစားထိုးရန်
- **Visual** (plain, line, or block): စာသား အစိတ်အပိုင်းများကို ရွေးချယ်ရန်
- **Command-line**: Command များကို Run ရန်

မူလအစတွင် Vim သည် Normal mode ၌ ရှိနေမည် ဖြစ်သည်။ မည်သည့် Mode မှမဆို Normal mode သို့ ပြန်လည် ရောက်ရှိရန် `<ESC>` (Escape key) ကို နှိပ်ရမည်။ 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)၊ နှင့် Command-line mode သို့ ဝင်ရောက်ရန် `:` တို့ကို နှိပ်ရမည် ဖြစ်သည်။

အခြေခံများ (Basics)

စာသား ထည့်သွင်းခြင်း (Inserting text)

Normal mode မှနေ၍ `i` ကို နှိပ်ပါက Insert mode သို့ ရောက်ရှိမည် ဖြစ်ပြီး အခြား Text Editor များကဲ့သို့ စာသားများ ရိုက်ထည့်နိုင်မည် ဖြစ်သည်။ `<ESC>` နှိပ်ပါက Normal mode သို့ ပြန်လည် ရောက်ရှိမည် ဖြစ်သည်။

Buffers, tabs, နှင့် windows

Vim သည် ဖွင့်လှစ်ထားသော ဖိုင်များကို “Buffers” ဟု ခေါ်ဆိုသည်။ Vim Session တစ်ခုတွင် Tab များစွာ ပါဝင်နိုင်ပြီး Tab တစ်ခုစီတွင် Window (ခွဲထားသော Pane) များစွာ ပါဝင်နိုင်သည်။ Window တစ်ခုစီသည် Buffer တစ်ခုကို ပြသပေးသည်။ Buffer တစ်ခုတည်းကို Window အများအပြားတွင် ပြိုင်တူ ဖွင့်လှစ် ကြည့်ရှုနိုင်ပါသည်။

Command-line

Normal mode တွင် `:` ရိုက်ထည့်ပါက Screen အောက်ခြေရှိ Command line သို့ ရောက်ရှိမည် ဖြစ်သည်-

- `:q` ထွက်ရန် (Window ပိတ်ရန်)
- `:w` ဖိုင် သိမ်းဆည်းရန် (“write”)
- `:wq` ဖိုင် သိမ်းဆည်းပြီး ထွက်ရန်
- `:e {name of file}` ပြင်ဆင်လိုသော ဖိုင်ကို ဖွင့်ရန်
- `:ls` ဖွင့်ထားသော Buffer များကို ပြသရန်
- `:help {topic}` အကူအညီ ဖွင့်ရန်

Vim Interface သည် ပရိုဂရမ်းမင်း ဘာသာစကားတစ်ခု ဖြစ်သည် (Vim's interface is a programming language)

လမ်းကြောင်းရှာခြင်း (Movement)

Normal mode တွင် သွားလာလှုပ်ရှားရန် Command များကို အသုံးပြုသည်-

- အခြေခံ ရွှေ့လျားမှု: `hjkl` (ဘယ်၊ အောက်၊ အထက်၊ ညာ)
- စကားလုံးများ: `w` (နောက် စကားလုံး)၊ `b` (စကားလုံး အစ)၊ `e` (စကားလုံး အဆုံး)
- စာကြောင်းများ: `0` (စာကြောင်း အစ)၊ `^` (ပထမဆုံး စာသား)၊ `$` (စာကြောင်း အဆုံး)
- ဖန်သားပြင်: `H` (အထက်)၊ `M` (အလယ်)၊ `L` (အောက်)
- Scroll: `Ctrl-u` (အထက်)၊ `Ctrl-d` (အောက်)
- ဖိုင်: `gg` (ဖိုင် အစ)၊ `G` (ဖိုင် အဆုံး)
- စာကြောင်း နံပါတ်: `:{number}<CR>` သို့မဟုတ် `{number}G`
- ရှာဖွေခြင်း: `/{regex}` ၊ match များအကြား သွားလာရန် `n` / `N`

စာသား ပြင်ဆင်မှုများ (Edits)

Vim ၏ ပြင်ဆင်မှု Command များမှာ-

- `i` Insert mode သို့ ဝင်ရောက်ရန်
- `o` / `O` အောက် / အထက် တွင် စာကြောင်းအသစ် ဖွင့်ရန်
- `d{motion}` {motion} အတိုင်း ဖျက်ရန် (ဥပမာ `dw` သည် စကားလုံး ဖျက်ရန်၊ `d$` သည် စာကြောင်း အဆုံးအထိ ဖျက်ရန်)
- `c{motion}` {motion} အတိုင်း ပြောင်းလဲရန် (ဥပမာ `cw` သည် စကားလုံး ပြောင်းလဲရန်)
- `x` စာလုံး တစ်လုံး ဖျက်ရန် (`dl` နှင့် တူညီသည်)
- `u` ပြန်ပြင်ရန် (undo)၊ `<C-r>` ပြန်လုပ်ရန် (redo)
- `y` စာသား ကူးယူရန် (yank)

- `p` စာသား ကူးထည့်ရန် (paste)

အရေအတွက် ပေါင်းစပ်ခြင်း (Counts)

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

Modifiers (အထူး ပြုပြင်မှုများ)

- `ci(` လက်ရှိ ကွင်းစကွင်းပိတ် `()` အတွင်းမှ စာသားကို ပြောင်းလဲရန်
- `ci[` လက်ရှိ လေးထောင့်ကွင်း `[]` အတွင်းမှ စာသားကို ပြောင်းလဲရန်
- `da'` Single quote အပါအဝင် အတွင်းမှ စာသားကို ဖျက်ရန်

Demo (လက်တွေ့ပြသမှု)

အောက်ပါ ဖျက်ဆီးထားသော **Fizz Buzz** (https://en.wikipedia.org/wiki/Fizz_buzz) ကုဒ်ကို Vim အသုံးပြု၍ ပြင်ဆင်ပြသထားသည်-

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

def main():
    fizz_buzz(10)
```

Vim ကို စိတ်ကြိုက် ပြင်ဆင်ခြင်း (Customizing Vim)

Vim ကို `~/.vimrc` ဖိုင် (Vimscript command များ ပါဝင်သော ဖိုင်) ဖြင့် စိတ်ကြိုက် ပြင်ဆင်နိုင်ပါသည်။ ကျွန်ုပ်တို့ ပြင်ဆင်ထားသော မူလ Config ကို [ဤနေရာတွင် (/2020/files/vimrc)](/2020/files/vimrc) **ဒေါင်းလုဒ်ဆွဲ၍ `~/.vimrc` အဖြစ် သိမ်းဆည်း အသုံးပြုနိုင်ပါသည်။**

Vim ကို တိုးချဲ့ခြင်း (Extending Vim)

Vim 8.0 မှ စတင်၍ Plugin များကို `~/.vim/pack/vendor/start/` Directory အတွင်း သို့ ထည့်သွင်း၍ တိုက်ရိုက် အသုံးပြုနိုင်ပါသည်။

ထင်ရှားသော Plugin အချို့မှာ- - **ctrlp.vim** (https://github.com/ctrlpvim/ctrlp.vim): ဖိုင်များ ရှာဖွေရန် - **ack.vim** (https://github.com/mileszs/ack.vim): ကုဒ် ရှာဖွေရန် - **nerdtree** (https://github.com/scrooloose/nerdtree): ဖိုင်တွဲများ ကြည့်ရှုရန် - **vim-easymotion** (https://github.com/easymotion/vim-easymotion): လျင်မြန်စွာ သွားလာရန်

အခြား ပရိုဂရမ်များတွင် Vim Mode အသုံးပြုခြင်း

- Bash တွင် `set -o vi`
- Zsh တွင် `bindkey -v`
- Environment variable: `export EDITOR=vim`

အဆင့်မြင့် Vim (Advanced Vim)

- Search and replace: `%s/foo/bar/g`
- Multiple windows: `:sp` / `:vsp`
- Macros: `q{character}` ဖြင့် Macro Record လုပ်ပြီး `@{character}` ဖြင့် ပြန်လည် Run ရန်။

အရင်းအမြစ်များ (Resources)

- `vimtutor` - Terminal တွင် `vimtutor` ဟု ရိုက်ထည့်၍ လေ့လာနိုင်သော သင်ခန်းစာ
- **Vim Adventures** (https://vim-adventures.com/) - Vim လေ့လာနိုင်သော ဂိမ်း
- **Vim Tips Wiki** (https://vim.fandom.com/wiki/Vim_Tips_Wiki)
- **Vi/Vim Stack Exchange** (https://vi.stackexchange.com/)

လေ့ကျင့်ခန်းများ (Exercises)

၁။ `vimtutor` ကို ပြီးစီးအောင် လေ့လာပါ။ ၂။ ကျွန်ုပ်တို့၏ **အခြေခံ vimrc** ကို ဒေါင်းလုဒ်လုပ်၍ `~/.vimrc` အဖြစ် သိမ်းဆည်း အသုံးပြုပါ။ ၃။ **ctrlp.vim** (https://github.com/ctrlpvim/ctrlp.vim) plugin ကို ထည့်သွင်း စမ်းသပ်ပါ။ ၄။ သင်ခန်းစာပါ **Demo** ကို မိမိ စက်ပေါ်တွင် ကိုယ်တိုင် ပြန်လည် ပြုလုပ်ကြည့်ပါ။ ၅။ ရှေ့လာမည့် ၁ လအတွင်း စာသား ပြင်ဆင်မှု အားလုံးအတွက် Vim ကို သီးသန့် အသုံးပြုပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. Visual Studio Code	https://code.visualstudio.com/	
2. Vim	https://www.vim.org/	
3. Fizz Buzz	https://en.wikipedia.org/wiki/Fizz_buzz	
4. ctrlp.vim	https://github.com/ctrlpvim/ctrlp.vim	
5. ack.vim	https://github.com/mileszs/ack.vim	
6. nerdtree	https://github.com/scrooloose/nerdtree	
7. vim-easymotion	https://github.com/easymotion/vim-easymotion	
8. Vim Adventures	https://vim-adventures.com/	
9. Vim Tips Wiki	https://vim.fandom.com/wiki/Vim_Tips_Wiki	
10. Vi/Vim Stack Exchange	https://vi.stackexchange.com/	

Metaprogramming

အနှစ်ချုပ် - Build systems, dependency management, testing, နှင့် continuous integration တို့ အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Metaprogramming

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=_Ms1Z4xfqv4

“metaprogramming” ဟု ပြောရာတွင် မည်သည့်အရာကို ဆိုလိုသနည်း။ Code ကို တိုက်ရိုက် ရေးသားခြင်း သို့မဟုတ် ပိုမို ထိရောက်စွာ အလုပ်လုပ်ခြင်းထက် *လုပ်ငန်းစဉ် (process)* နှင့် ပိုမို သက်ဆိုင်သော အရာများ၏ အစုအဝေးအတွက် ကျွန်ုပ်တို့ ပေးနိုင်သည့် အကောင်းဆုံး အမည်ဖြစ်ပါသည်။ ဤသင်ခန်းစာတွင် Code များကို Build လုပ်ခြင်း၊ Testing ပြုလုပ်ခြင်း နှင့် Dependency များကို စီမံခန့်ခွဲခြင်း စနစ်များကို လေ့လာသွားပါမည်။ ကျောင်းသားတစ်ယောက်၏ နေ့စဉ် ဘဝတွင် ဤသည်တို့မှာ အရေးပါမှု နည်းပါးသည် ဟု ထင်ရသော်လည်း အလုပ်သင် အဖြစ် သို့မဟုတ် အပြင် ကမ္ဘာစစ်စစ်တွင် ကြီးမားသော Code base များ နှင့် စတင် ထိတွေ့သည့်အခါ ဤအရာများကို နေရာတိုင်း၌ တွေ့မြင်ရမည် ဖြစ်ပါသည်။

“metaprogramming” ဟူသော ဝေါဟာရသည် “[ပရိုဂရမ်များကို စီမံဆောင်ရွက်သော ပရိုဂရမ်များ](https://en.wikipedia.org/wiki/Metaprogramming)” ဟုလည်း အဓိပ္ပာယ် ရနိုင်သည်ကို သတိပြုပါ၊ သို့သော် ဤသင်ခန်းစာ၏ ရည်ရွယ်ချက်မှာ ထို အဓိပ္ပာယ်မျိုး မဟုတ်ပါ။

Build systems

LaTeX ဖြင့် စာတမ်းတစ်စောင် ရေးသားပါက၊ စာတမ်း ထွက်ပေါ်လာရန် မည်သည့် Command များကို runရမည်နည်း။ Benchmark များကို runရန်၊ ပုံဆွဲရန်၊ ထိုပုံကို စာတမ်းထဲ ထည့်သွင်းရန် Command များရော မည်သို့နည်း။ သို့မဟုတ် အတန်းတွင် ပေးထားသော Code များကို Compile လုပ်ပြီး Test များကို runရန် မည်သို့ လုပ်ရမည်နည်း။

Code ပါသည်ဖြစ်စေ မပါသည်ဖြစ်စေ ပရောဂျက် အများစုတွင် “build process” (ဆောက်လုပ်မှု လုပ်ငန်းစဉ်) တစ်ခု ရှိကြပါသည်။ Input များကိုနှာပြင်မှ Output သို့ ရောက်ရှိရန် လုပ်ဆောင်ရမည့် လုပ်ငန်းစဉ် အစဉ်လိုက် ဖြစ်ပါသည်။ မကြာခဏဆိုသလို ထို လုပ်ငန်းစဉ်တွင် အဆင့်များစွာ၊ လမ်းခွဲများစွာ ပါဝင်နိုင်ပါသည်။ ပုံဆွဲရန် ဤသည်ကို runပါ၊ ရလဒ်များ ထုတ်ရန် ထိုသည်ကို runပါ၊ စာတမ်းအချောသတ် ထုတ်ရန် အခြားတစ်ခု runပါ ဟူ၍ ဖြစ်ပါသည်။ ဤအတန်းတွင် တွေ့မြင်ခဲ့ရသော အရာများစွာကဲ့သို့ပင် ဤ စိတ်ပျက်ဖွယ်ရာများနှင့် ကြုံတွေ့ရသူမှာ သင် တစ်ယောက်တည်း မဟုတ်ပါ၊ ကံကောင်းစွာပင် သင့်ကို ကူညီနိုင်မည့် Tool များစွာ ရှိနေပါပြီ။

ယင်းတို့ကို အများအားဖြင့် “build systems” ဟု ခေါ်ဆိုကြပြီး အမျိုးအစား *များစွာ* ရှိကြပါသည်။ မည်သည့် စနစ်ကို သုံးမည်ဆိုသည်မှာ လက်ရှိ လုပ်ဆောင်ရမည့် အလုပ်၊ နှစ်သက်သော ဘာသာစကား နှင့် ပရောဂျက် ဆိုဒ် ပေါ်မူတည်ပါသည်။ သို့သော် ယင်းတို့၏ ပင်မ အနှစ်သာရမှာ အလွန် ဆင်တူကြပါသည်။ အချို့သော *dependencies* များ၊ *targets* များ နှင့် တစ်ခုမှ တစ်ခုသို့ ကူးပြောင်းရန် *rules* များကို သတ်မှတ်ပေးရသည်။ Build system ကို သီးခြား Target တစ်ခု လိုချင်ကြောင်း ပြောလိုက်ပါက၊ ယင်း Target ၏ မှီခိုမှုများ အားလုံးကို ရှာဖွေပြီး နောက်ဆုံး Target မထွက်မချင်း ရလဒ်အလယ်အလတ်များကို ထုတ်လုပ်ရန် Rule များကို ကျင့်သုံးပေးခြင်းမှာ ယင်း၏ အလုပ် ဖြစ်ပါသည်။ ထို့အပြင် Build system သည် Dependency မပြောင်းလဲဘော ယခင် Build မှ ရလဒ် ရရှိနိုင်သော Target များအတွက် မလိုအပ်ဘဲ Rule များကို runမနေဘဲ ထိရောက်စွာ ဆောင်ရွက်ပေးပါသည်။

`make` သည် အသုံးအများဆုံး Build system များထဲမှ တစ်ခု ဖြစ်ပြီး UNIX အခြေပြု ကွန်ပျူတာ အများစုတွင် တပ်ဆင်ထားသည်ကို တွေ့ရမည် ဖြစ်သည်။ အားနည်းချက် အချို့ ရှိသော်လည်း ရိုးရှင်းမှ အသင့်အတင့် ပရောဂျက်များအတွက် အလွန် အဆင်ပြေပါသည်။ `make` ကို runလိုက်သောအခါ လက်ရှိ Directory ရှိ `Makefile` ဟုခေါ်သော ဖိုင်ကို ဖတ်ရှုပါသည်။ Target များ၊ Dependencies များနှင့် Rules များ အားလုံးကို ထိုဖိုင်ထဲတွင် သတ်မှတ်ထားသည်။ အောက်ပါ နမူနာကို ကြည့်ပါ -

```
paper.pdf: paper.tex plot-data.png
    pdflatex paper.tex

plot-%.png: %.dat plot.py
    ./plot.py -i $*.dat -o $@
```

ဤဖိုင်ရှိ ညွှန်ကြားချက် တစ်ခုစီသည် ညာဘက်ခြမ်းကို အသုံးပြု၍ ဘယ်ဘက်ခြမ်းကို မည်သို့ ထုတ်လုပ်မည် ဟူသော Rule ဖြစ်ပါသည်။ သို့မဟုတ် အခြားတစ်မျိုး ပြောရလျှင် ညာဘက်ခြမ်းရှိ အရာများသည် Dependencies များ ဖြစ်ကြပြီး ဘယ်ဘက်ခြမ်းသည် Target ဖြစ်ပါသည်။ ခြားထားသော အောက်ပါ Block သည် ထို Dependency များမှ Target ကို ထုတ်လုပ်ပေးမည့် ပရိုဂရမ် အစဉ်လိုက် ဖြစ်သည်။ `make` တွင် ပထမဆုံး ညွှန်ကြားချက်သည် Default goal လည်း ဖြစ်ပါသည်။ Argument မပါဘဲ `make` runပါက ဤ Target ကို Build လုပ်မည် ဖြစ်သည်။ သို့မဟုတ် `make plot-data.png` ဟု runပါက ထို Target ကို သီးသန့် Build လုပ်မည် ဖြစ်သည်။

Rule ရှိ `%` သည် “pattern” ဖြစ်ပြီး ဘယ်ဘက်နှင့် ညာဘက်ရှိ တူညီသော String ကို ကိုက်ညီစေပါသည်။ ဥပမာ Target `plot-foo.png` ကို တောင်းဆိုပါက `make` သည် Dependency များ ဖြစ်သော `foo.dat` နှင့် `plot.py` တို့ကို ရှာဖွေမည် ဖြစ်သည်။ ယခု အလွတ် ဖြစ်နေသော Directory တွင် `make` runပါက မည်သို့ ဖြစ်မည်ကို ကြည့်ပါ -

```
$ make
make: *** No rule to make target 'paper.tex', needed by 'paper.pdf'.  Stop.
```

`make` က `paper.pdf` ကို Build ရန် `paper.tex` လိုအပ်ကြောင်းနှင့် ထိုဖိုင် ဖန်တီးရန် Rule မရှိကြောင်း သတိပေးလိုက်ခြင်း ဖြစ်သည်။ ဖိုင်ကို ဖန်တီးကြည့်ကြပါစို့ -

```
$ touch paper.tex
$ make
make: *** No rule to make target 'plot-data.png', needed by 'paper.pdf'.  Stop.
```

စိတ်ဝင်စားစရာမှာ `plot-data.png` ကို ဖန်တီးရန် Rule *ရှိသော်လည်း* ယင်းသည် Pattern rule ဖြစ်နေခြင်း ဖြစ်သည်။ Source file (`data.dat`) မရှိသေးသောကြောင့် `make` က ထိုဖိုင်ကို မထုတ်လုပ်နိုင်ကြောင်း ပြောခြင်း ဖြစ်သည်။ ဖိုင်များ အားလုံး ဖန်တီးကြည့်ကြပါစို့ -

```
$ cat paper.tex
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\includegraphics[scale=0.65]{plot-data.png}
\end{document}
$ cat plot.py
#!/usr/bin/env python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('-i', type=argparse.FileType('r'))
parser.add_argument('-o')
args = parser.parse_args()

data = np.loadtxt(args.i)
plt.plot(data[:, 0], data[:, 1])
plt.savefig(args.o)
$ cat data.dat
1 1
2 2
3 3
4 4
5 8
```

ယခု `make` runလိုက်ပါက မည်သို့ ဖြစ်မည်နည်း -

```
$ make
./plot.py -i data.dat -o plot-data.png
pdflatex paper.tex
... lots of output ...
```

PDF ဖိုင် ထွက်ရှိလာသည်ကို တွေ့ရမည် ဖြစ်သည်! `make` ကို ထပ်မံ runကြည့်ပါက မည်သို့ ဖြစ်မည်နည်း -

```
$ make
make: 'paper.pdf' is up to date.
```

ဘာမှ ထပ်မလုပ်တော့ပါ! အဘယ်ကြောင့်နည်း။ အကြောင်းမှာ ထပ်မလုပ်ရန် မလိုအပ်သောကြောင့် ဖြစ်သည်။ ယခင်က Build လုပ်ထားသော Target များသည် မူလ Dependency များနှင့် ယှဉ်လျှင် အပ်ဒိတ် ဖြစ်နေဆဲ ဖြစ်ကြောင်း စစ်ဆေးလိုက်ခြင်း ဖြစ်သည်။ `paper.tex` ကို ပြင်ဆင်ပြီး `make` ပြန်runကြည့်ပါက -

```
$ vim paper.tex
$ make
pdflatex paper.tex
...
```

`make` သည် `plot.py` ကို ပြန်လည် မrunခဲ့သည်ကို သတိပြုပါ၊ အကြောင်းမှာ မလိုအပ်သောကြောင့် ဖြစ်သည်; `plot-data.png` ၏ Dependency မည်သည့်အရာမျှ မပြောင်းလဲခဲ့ပါ!

Dependency management

ပိုမို ကျယ်ပြန့်သော အဆင့်တွင် သင်၏ Software ပရောဂျက်များသည် အခြား ပရောဂျက်များကို Dependency အဖြစ် မှီခိုနေလေ့ ရှိပါသည်။ တပ်ဆင်ထားသော ပရိုဂရမ်များ (ဥပမာ `python`)၊ System package များ (ဥပမာ `openssl`) သို့မဟုတ် သင်၏ ဘာသာစကားရှိ Library များ (ဥပမာ `matplotlib`) ကို မှီခိုနိုင်ပါသည်။ ယနေ့ခေတ်တွင် Dependency အများစုကို တစ်နေရာတည်း၌ ထားရှိပြီး လွယ်ကူစွာ တပ်ဆင်နိုင်သော *repository* များမှ ရရှိနိုင်ပါသည်။ Ubuntu system package များအတွက် `apt` tool မှတဆင့် ဝင်ရောက်သော Ubuntu package repositories၊ Ruby library များအတွက် RubyGems၊ Python library များအတွက် PyPI သို့မဟုတ် Arch Linux အတွက် Arch User Repository တို့ ဖြစ်ကြပါသည်။

ဤ Repository များနှင့် ချိတ်ဆက် ဆောင်ရွက်သည့် နည်းလမ်းများသည် Repository တစ်ခုနှင့်တစ်ခု Tool တစ်ခုနှင့်တစ်ခု ကွဲပြားသောကြောင့် သီးခြား တစ်ခုချင်းစီ၏ အသေးစိတ်ကို သွားမည် မဟုတ်ပါ။ အစားယင်းတို့ အားလုံး သုံးစွဲကြသော အသုံးများသော ဝေါဟာရများကို သင်ကြားပေးပါမည်။ ပထမဆုံးမှာ *versioning* (မူကွဲ သတ်မှတ်ခြင်း) ဖြစ်ပါသည်။ အခြား ပရောဂျက်များ မှီခိုသော ပရောဂျက် အများစုသည် Release တိုင်းတွင် *version number* တစ်ခု ထုတ်ပြန်ကြသည်။ ဥပမာ `8.1.3` သို့မဟုတ် `64.1.20192004`။ ကိန်းဂဏန်းများ ဖြစ်လေ့ရှိကြသည်။ Version number များသည် ရည်ရွယ်ချက် အများအပြား ဆောင်ရွက်ပြီး အရေးကြီးဆုံးမှာ Software ဆက်လက် အလုပ်လုပ်စေရန် ဖြစ်သည်။ ဥပမာ ကျွန်ုပ်၏ Library တွင် Function တစ်ခု၏ အမည်ကို ပြောင်းလဲလိုက်သော Release အသစ်တစ်ခု ထုတ်လိုက်သည် ဆိုပါစို့။ အကယ်၍ အခြားသူက ကျွန်ုပ်၏ Library ကို မှီခို၍ Build လုပ်ပါက Function မရှိတော့သဖြင့် Build ပျက်စီးသွားနိုင်ပါသည်။ Versioning သည် သီးခြား Version သို့မဟုတ် Version အပိုင်းအခြားကို သတ်မှတ်ခွင့် ပြုခြင်းဖြင့် ဤပြဿနာကို ဖြေရှင်းပေးပါသည်။

သို့သော် ဤသည်မှာလည်း အပြည့်အဝ မဟုတ်သေးပါ! အကယ်၍ Public API ကို မပြောင်းလဲဘဲ လုံခြုံရေး အပ်ဒိတ်တစ်ခု ထုတ်လိုက်ပြီး မူလ Version အသုံးပြုသူ အားလုံး ချက်ချင်း သုံးသင့်သည် ဆိုပါစို့။ ဤနေရာတွင် Version နံပါတ် အုပ်စုများ ရောက်ရှိလာပါသည်။ နံပါတ်တစ်ခုစီ၏ အဓိပ္ပာယ်သည် ပရောဂျက်အလိုက် ကွဲပြားသော်လည်း အသုံးများသော စံနှုန်းတစ်ခုမှာ ***semantic versioning*** (https://semver.org/) ဖြစ်ပါသည်။ Semantic versioning တွင် Version နံပါတ်တိုင်းသည် major.minor.patch ပုံစံ ဖြစ်သည် -

- API ပြောင်းလဲမှု မရှိပါက patch version ကို တိုးပါ။
- နောက်ပြန် ညီညွတ်မှု ရှိသော (backwards-compatible) API အသစ် ထည့်ပါက minor version ကို တိုးပါ။
- နောက်ပြန် ညီညွတ်မှု မရှိသော API ပြောင်းလဲမှု ပြုလုပ်ပါက major version ကို တိုးပါ။

ဤသည်မှာ အကျိုးကျေးဇူးများစွာ ပေးစွမ်းပါသည်။ ယခုအခါ ကျွန်ုပ်၏ ပရောဂျက်သည် သင်၏ ပရောဂျက်ကို မှီခိုပါက၊ Major version တူညီနေသရွေ့ နောက်ဆုံး Release ကို အသုံးပြုခြင်းသည် ဘေးကင်းသင့်ပါသည်။ ဥပမာ Version `1.3.7` ကို မှီခိုပါက `1.3.8`, `1.6.1` သို့မဟုတ် `1.3.0` တို့ဖြင့် Build လုပ်ခြင်းသည် အဆင်ပြေသင့်ပါသည်။ Version `2.2.4` မှ Major version တိုးသွားသဖြင့် အဆင်မပြေနိုင်ပါ။ Python ၏ Version နံပါတ်များတွင်လည်း ဤသည်ကို တွေ့နိုင်ပါသည်။ Python 2 နှင့် Python 3 code များ ရောနှော၍ မရခြင်းမှာ *major* version bump ဖြစ်ခဲ့သောကြောင့် ဖြစ်သည်။

Dependency စီမံခန့်ခွဲမှု စနစ်များနှင့် အလုပ်လုပ်သည့်အခါ *lock files* အယူအဆကိုလည်း တွေ့ရနိုင်ပါသည်။ Lock file ဆိုသည်မှာ လက်ရှိ မှီခိုနေသော Dependency တစ်ခုစီ၏ တိကျသော Version ကို စာရင်းပြုလုပ်ထားသည့် ဖိုင်ဖြစ်ပါသည်။ သာမန်အားဖြင့် အပ်ဒိတ် ပရိုဂရမ်ကို runမှသာ Version အသစ်များသို့ အဆင့်မြှင့်မည် ဖြစ်သည်။ မလိုအပ်ဘဲ ပြန်လည် Compile လုပ်ခြင်းကို ရှောင်ရှားရန်၊ ပြန်လည် ထုတ်လုပ်နိုင်သော Build များ ရရှိရန် စသည့် အကြောင်းပြချက်များ ရှိပါသည်။ ဤ Lock ပြုလုပ်ခြင်း၏ ပြင်းထန်သော မူကွဲမှာ *vendoring* ဖြစ်ပြီး ယင်းသည် Dependency များ၏ Code အားလုံးကို မိမိ ပရောဂျက်ထဲသို့ ကူးယူလိုက်ခြင်း ဖြစ်သည်။ ၎င်းသည် အပြောင်းအလဲများကို အပြည့်အဝ ထိန်းချုပ်နိုင်စေသော်လည်း Maintainer များထံမှ အပ်ဒိတ်များကို ကိုယ်တိုင် ယူဆောင်ရမည် ဖြစ်သည်။

Continuous integration systems

ကြီးမားသော ပရောဂျက်များတွင် အလုပ်လုပ်လာသည်နှင့်အမျှ အပြောင်းအလဲ ပြုလုပ်တိုင်း ထပ်မံ လုပ်ဆောင်ရမည့် အလုပ်များ ရှိလာသည်ကို တွေ့ရမည်။ Documentation မူကွဲအသစ် တင်ခြင်း၊ Compiled version တင်ခြင်း၊ Code များကို PyPI သို့ လွှင့်ခြင်း၊ Test စာအုပ်များ runခြင်း စသည်တို့ ဖြစ်သည်။ GitHub တွင် Pull Request တင်တိုင်း စတိုင် စစ်ဆေးစေချင်သလား။ ဤကဲ့သို့သော လိုအပ်ချက်များအတွက် Continuous Integration ကို လေ့လာရမည် ဖြစ်သည်။

Continuous integration (CI) ဆိုသည်မှာ “Code ပြောင်းလဲတိုင်း runမည့် အရာများ” အတွက် ခြုံငုံ ခေါ်ဆိုသော ဝေါဟာရ ဖြစ်ပြီး Open-source ပရောဂျက်များအတွက် အခမဲ့ ပေးသော CI ကုမ္ပဏီများစွာ ရှိပါသည်။ Travis CI, Azure Pipelines နှင့် GitHub Actions တို့ ဖြစ်ကြသည်။ ယင်းတို့ အားလုံးသည် ပရောဂျက်တွင် ဖိုင်တစ်ခု ထည့်သွင်း၍ ဖြစ်ရပ်များ ဖြစ်ပေါ်သည့်အခါ မည်သို့ ပြုလုပ်ရမည်ကို သတ်မှတ်ပေးရသည်။ အသုံးအများဆုံး Rule မှာ “Code ပို့လိုက်ပါက Test များ runပါ” ဟူ၍ ဖြစ်သည်။ ဖြစ်ရပ် ဖြစ်ပေါ်ပါက Virtual machine ကို စတင်၍ ရလဒ်များကို မှတ်တမ်းတင်ပေးပါသည်။ Test ပျက်စီးပါက အကြောင်းကြားစာ ရရှိရန် ပြုလုပ်ထားနိုင်ပါသည်။

CI စနစ်၏ ဥပမာတစ်ခုအဖြစ် ဤအတန်း၏ ဝဘ်ဆိုက်ကို GitHub Pages ဖြင့် ပြင်ဆင်ထားပါသည်။ Pages သည် `master` သို့ Push လုပ်တိုင်း Jekyll ကို runပေးသော CI Action ဖြစ်ပါသည်။ ဤသည်မှာ ဝဘ်ဆိုက်ကို အပ်ဒိတ်လုပ်ရန် အလွန် လွယ်ကူစေပါသည်! Local တွင် ပြင်ဆင်သည်၊ Git ဖြင့် Commit လုပ်သည်၊ Push လုပ်သည်။ CI က ကျန်သည်များကို လုပ်ဆောင်ပေးပါသည်။

A brief aside on testing

ကြီးမားသော Software ပရောဂျက် အများစုတွင် “test suite” ပါရှိကြပါသည်။ Testing ၏ အယူအဆအချို့ကို အောက်ပါအတိုင်း တွေ့ရနိုင်ပါသည် -

- Test suite: Test အားလုံး၏ စုစည်းမှု
- Unit test: သီးခြား အင်္ဂါရုပ်တစ်ခုကို သီးသန့် စမ်းသပ်သော “micro-test”
- Integration test: အစိတ်အပိုင်း အသီးသီး အတူတကွ အလုပ်လုပ်ပုံကို စမ်းသပ်သော “macro-test”
- Regression test: ယခင်က ဖြစ်ပွားခဲ့သော Bug ထပ်မံ မဖြစ်စေရန် စမ်းသပ်သော Test
- Mocking: သီးခြား အင်္ဂါရုပ်များကို သီးသန့် စမ်းသပ်နိုင်ရန် Function, Module များကို အတု ပြုလုပ်ခြင်း (ဥပမာ “mock the network” သို့မဟုတ် “mock the disk”)

Exercises

1. Makefile အများစုတွင် `clean` ဟုခေါ်သော Target ပါရှိသည်။ ယင်းသည် `clean` ဖိုင် ထုတ်ရန် မဟုတ်ဘဲ ပြန်လည် Build လုပ်နိုင်သော ဖိုင်များကို ရှင်းလင်းရန် ဖြစ်သည်။ `paper.pdf` ၏ `Makefile` တွင် `clean` target ထည့်သွင်းပါ။ Target ကို **phony** (https://www.gnu.org/software/make/manual/html_node/Phony-Targets.html) ပြုလုပ်ရန် လိုပါမည်။ **`git ls-files`** (https://git-scm.com/docs/git-ls-files) ကို သုံးနိုင်ပါသည်။

2. **Rust ၏ build system** (https://doc.rust-lang.org/cargo/reference/specifying-dependencies.html) ရှိ Dependency Version သတ်မှတ်ချက် နည်းလမ်းများကို လေ့လာပါ။ နည်းလမ်း တစ်ခုစီအတွက် သင့်တော်သော အခြေအနေများကို စဉ်းစားပါ။

3. Git ကိုယ်တိုင်သည် ရိုးရှင်းသော CI စနစ်အဖြစ် အလုပ်လုပ်နိုင်ပါသည်။ Git repo တိုင်းရှိ `.git/hooks` တွင် သီးခြား အရေးယူမှုများ ဖြစ်ပေါ်ပါက runမည့် Script ဖိုင်များ ရှိပါသည်။ `make paper.pdf` runမည့် **`pre-commit`** (https://git-scm.com/docs/githooks#_pre_commit) hook တစ်ခု ရေးပါ၊ `make` မ အောင်မြင်ပါက Commit ကို ငြင်းပယ်ပါစေ။

4. **GitHub Pages** (https://pages.github.com/) ဖြင့် အလိုအလျောက် ထုတ်ဝေသော စာမျက်နှာတစ်ခု ပြင်ဆင်ပါ။ Repo ရှိ Shell ဖိုင်များတွင် `shellcheck` runရန် **GitHub Action** (https://github.com/features/actions) ထည့်သွင်းပါ။

5. Repo ရှိ `.md` ဖိုင်များတွင် **`proselint`** (https://github.com/amperser/proselint) သို့မဟုတ် **`write-good`** (https://github.com/btford/write-good) runမည့် မိမိပိုင် GitHub action ကို ဖန်တီးပါ (https://help.github.com/en/actions/automating-your-workflow-with-github-actions/building-actions)။ စာလုံးပေါင်းများသော PR ဖြင့် စမ်းသပ်ပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

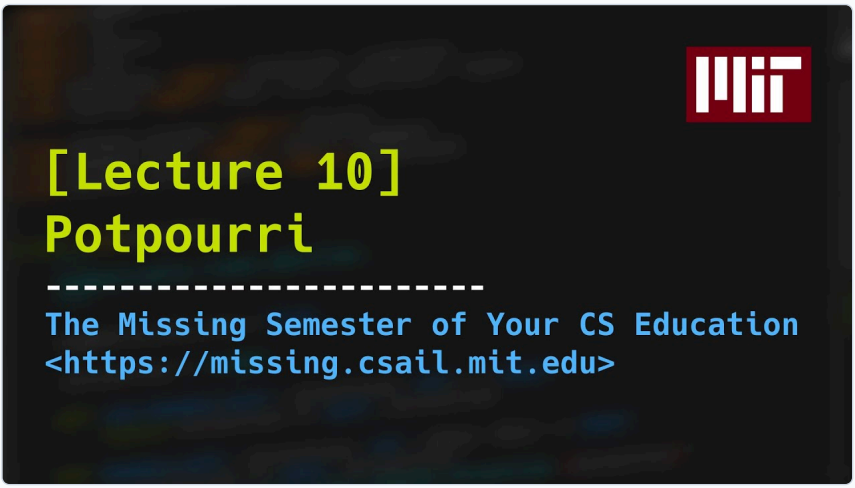
Resource / Description	URL	Micro QR
1. ပရိုဂရမ်များကို စိမံဆောင်ရွက်သော ပရိုဂရမ်များ	https://en.wikipedia.org/wiki/Metaprogramming	
2. _semantic versioning_	https://semver.org/	
3. phony	https://www.gnu.org/software/make/manual/html_node/Phony-Targets.html	
4. `git ls-files`	https://git-scm.com/docs/git-ls-files	
5. Rust ၏ build system	https://doc.rust-lang.org/cargo/reference/specifying-dependencies.html	
6. `pre-commit`	https://git-scm.com/docs/githooks#_pre_commit	
7. GitHub Pages	https://pages.github.com/	
8. GitHub Action	https://github.com/features/actions	
9. `proselint`	https://github.com/amperser/proselint	
10. `write-good`	https://github.com/btford/write-good	
11. ဖန်တီးပါ	https://help.github.com/en/actions/automating-your-workflow-with-github-actions/building-actions	

အထွေထွေ ခေါင်းစဉ်များ (Potpourri)

အနှစ်ချုပ် - Keyboard remapping, daemons, backups, APIs အပါအဝင် အသုံးဝင်သော ခေါင်းစဉ် အမျိုးအစားများစွာ အကြောင်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

အထွေထွေ ခေါင်းစဉ်များ (Potpourri)



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=JZDt-PRq0uo

Table of Contents

Keyboard remapping

Programmer တစ်ယောက်အဖြစ် သင်၏ ကီးဘုတ်သည် အဓိက Input နည်းလမ်း ဖြစ်ပါသည်။ ကွန်ပျူတာရှိ အရာအားလုံးကဲ့သို့ပင် ယင်းကို စိတ်ကြိုက် ပြင်ဆင်ဖန်တီးနိုင်ပါသည် (ပြင်ဆင်ရန်လည်း ထိုက်တန်ပါသည်)။

အခြေခံကျသော အပြောင်းအလဲမှာ ခလုတ်များကို ပြန်လည် သတ်မှတ်ခြင်း (remap) ဖြစ်ပါသည်။ ဤသည်မှာ သီးခြား ခလုတ်တစ်ခုကို နှိပ်လိုက်သည့်အခါ ယင်းကို ဖမ်းယူပြီး အခြား ခလုတ်တစ်ခုဖြင့် အစားထိုးပေးသည့် Software ဖြင့် ပြုလုပ်လေ့ ရှိသည်။ ဥပမာ - - Caps Lock ကို Ctrl သို့မဟုတ် Escape သို့ Remap ပြုလုပ်ခြင်း။ Caps Lock သည် အဆင်ပြေသော နေရာတွင် ရှိသော်လည်း မကြာခဏ မသုံးသောကြောင့် ဤသို့ သတ်မှတ်ရန် အလွန် အကြံပြုပါသည်။ - PrtSc ကို သီချင်း Play/Pause သို့ Remap ပြုလုပ်ခြင်း။ - Ctrl နှင့် Meta (Windows သို့မဟုတ် Command) ခလုတ်များကို လဲလှယ်ခြင်း။

ထို့အပြင် ခလုတ်များကို မိမိ စိတ်ကြိုက် Command များသို့လည်း တွဲဆက်နိုင်ပါသည်။ - Terminal သို့မဟုတ် Browser window အသစ် ဖွင့်ခြင်း။ - သီးခြား စာသားများ ရိုက်နှိပ်ပေးခြင်း (ဥပမာ သင်၏ Email လိပ်စာ သို့မဟုတ် ကျောင်းသား ID နံပါတ်)။ - ကွန်ပျူတာ သို့မဟုတ် မျက်နှာပြင်ကို Sleep ပြုလုပ်ခြင်း။

ပိုမို ရှုပ်ထွေးသော ပြင်ဆင်မှုများလည်း ပြုလုပ်နိုင်ပါသည် - - Shift ကို ၅ ကြိမ် နှိပ်ပါက Caps Lock ဖွင့်/ပိတ် ပြုလုပ်ခြင်း။ - ခလုတ်ကို မြန်မြန် နှိပ်ခြင်း (tap) နှင့် ဖိထားခြင်း (hold) အပေါ် မူတည်၍ Remap ပြုလုပ်ခြင်း (ဥပမာ Caps Lock ကို မြန်မြန် နှိပ်ပါက Esc ဖြစ်ပြီး ဖိထားပါက Ctrl ဖြစ်စေခြင်း)။

လေ့လာရန် Software အချို့ - - macOS - **karabiner-elements** (https://karabiner-elements.pqrs.org/), **skhd** (https://github.com/koekeishiya/skhd) သို့မဟုတ် **BetterTouchTool** (https://folivora.ai/) - Linux - **xmodmap** (https://wiki.archlinux.org/index.php/Xmodmap) သို့မဟုတ် **Autokey** (https://github.com/autokey/autokey) - Windows - Control Panel တွင် မူလပါရှိသည်၊ **AutoHotkey** (https://www.autohotkey.com/) သို့မဟုတ် **SharpKeys** (https://www.randyrants.com/category/sharpkeys/) - QMK - ကီးဘုတ်က Custom firmware ကို ထောက်ပံ့ပါက **QMK** (https://docs.qmk.fm/) ကို သုံး၍ Hardware ပေါ်တွင် တိုက်ရိုက် သတ်မှတ်နိုင်ပါသည်။

Daemons

Daemons အယူအဆကို စာလုံးသစ် ဖြစ်နေသည့်တိုင် သတိပြုမိပေမည်။ ကွန်ပျူတာ အများစုတွင် User က စတင် runသည်ကို မစောင့်ဘဲ Background တွင် အမြဲတမ်း runနေသော Process များ ရှိကြပါသည်။ ဤ Process များကို Daemons ဟု ခေါ်ဆိုပြီး runနေသော ပရိုဂရမ် အမည်များသည် အဆုံးတွင် `d` ဖြင့် ဆုံးလေ့ ရှိကြသည်။ ဥပမာ SSH daemon ဖြစ်သော `sshd` သည် SSH request များကို နားထောင်ပြီး Remote user တွင် လော့ဂ်အင် ဝင်ရန် Credentials ရှိမရှိ စစ်ဆေးပေးသော ပရိုဂရမ် ဖြစ်ပါသည်။

Linux တွင် `systemd` (system daemon) သည် Daemon process များကို စီမံရန် အသုံးအများဆုံး ဖြေရှင်းချက် ဖြစ်သည်။ runနေသော Daemon စာရင်းကို ကြည့်ရန် `systemctl status` ကို runနိုင်ပါသည်။ `systemctl` command ဖြင့် Service များကို `enable`, `disable`, `start`, `stop`, `restart` သို့မဟုတ် `status` စစ်ဆေးနိုင်ပါသည်။

`systemd` တွင် Daemon အသစ်များ ပြင်ဆင်သတ်မှတ်ရန် လွယ်ကူသော Interface ရှိပါသည်။ အောက်တွင် Python app အတွက် Daemon နမူနာ ဖြစ်ပါသည် -

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My Custom App
After=network.target

[Service]
User=foo
Group=foo
WorkingDirectory=/home/foo/projects/mydaemon
ExecStart=/usr/bin/local/python3.7 app.py
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

သီးခြား ကြိမ်နှုန်းဖြင့် ပရိုဂရမ် runလိုပါက Daemon အသစ် ဖန်တီးစရာ မလိုဘဲ စနစ်တွင် ပါဝင်ပြီးသား ဖြစ်သော `cron` (https://www.man7.org/linux/man-pages/man8/cron.8.html) ကို သုံးနိုင်ပါသည်။

FUSE

ခေတ်မီ Software စနစ်များကို အစိတ်အပိုင်း အသေးများဖြင့် ပေါင်းစပ် ဖွဲ့စည်းထားပါသည်။ သင်၏ Operating system တွင် Filesystem မတူညီသည်များကို သုံးနိုင်ခြင်းမှာ သုံးစွဲနိုင်သော Operation များ အတွက် တူညီသော ဘာသာစကား ရှိသောကြောင့် ဖြစ်သည်။

FUSE (https://en.wikipedia.org/wiki/Filesystem_in_Userspace) (Filesystem in User Space) သည် Filesystem များကို User program ဖြင့် အကောင်အထည်ဖော်ခွင့် ပြုပါသည်။ အလက်တွေ့တွင် User သည် စိတ်ကြိုက် လုပ်ဆောင်ချက်များကို Filesystem တွင် ရေးသားနိုင်ပါသည်။

FUSE Filesystem ၏ စိတ်ဝင်စားဖွယ် နမူနာများမှာ - - **sshfs** (https://github.com/libfuse/sshfs) - SSH မှတဆင့် Remote ဖိုင်များကို Local တွင် ဖွင့်ခြင်း။ - **rclone** (https://rclone.org/commands/rclone_mount/) - Dropbox, GDrive, Amazon S3, Google Cloud Storage တို့ကို Local တွင် Mount လုပ်ခြင်း။ - **gocryptfs** (https://nuetzlich.net/gocryptfs/) - Encrypted overlay system။ - **kbfs** (https://keybase.io/docs/kbfs) - End-to-end encryption ပါသော Distributed filesystem။ - **borgbackup** (https://borgbackup.readthedocs.io/en/stable/usage/mount.html) - Encrypted backup များကို လွယ်ကူစွာ ကြည့်နိုင်ရန် Mount လုပ်ခြင်း။

Backups

Backup မလုပ်ထားသော Data မည်သည့်အရာမဆို မည်သည့်အချိန်မဆို ထာဝစဉ် ပျောက်ကွယ်သွားနိုင်ပါသည်။ Data ကူးယူရသည်မှာ လွယ်ကူသော်လည်း ယုံကြည်စိတ်ချရသော Backup ထားရှိရန် ခက်ခဲပါသည်။

ပထမဦးစွာ Disc တစ်ခုတည်းတွင် Copy ကူးထားခြင်းသည် Backup မဟုတ်ပါ၊ အကြောင်းမှာ Disc ပျက်စီးပါက Data အားလုံး ပျောက်ကွယ်နိုင်သောကြောင့် ဖြစ်သည်။ အလားတူ အပြင်ဘက် Disc ကို အိမ်တွင် ထားခြင်းသည်လည်း မီးလောင်ခြင်း/အခိုးခံရခြင်းတို့ ကြုံနိုင်သဖြင့် အိမ်ပြင်ပ (off-site) တွင် Backup ထားရှိရန် အကြံပြုပါသည်။

Sync လုပ်သော နည်းလမ်းများ (Dropbox/GDrive) သို့မဟုတ် Disc mirroring (RAID) တို့သည် Backup မဟုတ်ပါ။ Data ဖျက်မိပါက သို့မဟုတ် Ransomware ထိပါက Synchronize ဖြစ်သွားမည် ဖြစ်သည်။

ကောင်းမွန်သော Backup ၏ အဓိက အင်္ဂါရုပ်များမှာ Versioning, Deduplication နှင့် Security တို့ ဖြစ်ကြသည်။ အသေးစိတ်အတွက် 2019 ခုနှစ်၏ **Backups သင်ခန်းစာ** (https://missing-semester-my.github.io/2019/backups) ကို ကြည့်ပါ။

APIs

အွန်လိုင်းရှိ ဝဘ်ဆိုက်/Service အများစုတွင် ယင်းတို့၏ Data များကို ပရိုဂရမ် နည်းလမ်းဖြင့် ဝင်ရောက်ကြည့်ရှုနိုင်သော “APIs” များ ရှိကြပါသည်။

API အများစုသည် သပ်ရပ်သော URL ပုံစံ ရှိကြပြီး၊ အများအားဖြင့် `api.service.com` တွင် တည်ရှိကြပါသည်။ တုံ့ပြန်မှုများကို JSON ဖြင့် Format ပြုလုပ်ထားလေ့ရှိပြီး `jq` (https://stedolan.github.io/jq/) ကဲ့သို့သော Tool ဖြင့် Parse ပြုလုပ်နိုင်ပါသည်။

အချို့သော API များသည် Authentication လိုအပ်ပြီး Secret *token* ပါဝင်ရသည်။ “**OAuth** (https://www.oauth.com/)” Protocol ကို အသုံးများပါသည်။

IFTTT (https://ifttt.com/) သည် API များကို ဗဟိုပြုထားသော ဝဘ်ဆိုက် ဖြစ်ပြီး ဝဘ်ဆိုက်များစွာ၏ ဖြစ်ရပ်များကို ချိတ်ဆက်ပေးသည်။

Common command-line flags/patterns

Command-line tool များတွင် အသုံးများသော Flag များမှာ -

- `--help` : အတိုချုပ် ညွှန်ကြားချက်များ ပြရန်။
- “dry run” flag: ပြောင်းလဲမည့် အရာများကို runမထားဘဲ ပြသပေးရန်။
- `--version` သို့မဟုတ် `-V` : Version နံပါတ် ပြရန်။
- `--verbose` သို့မဟုတ် `-v` : အသေးစိတ် Output များ ပြရန်။
- `-` : File name နေရာတွင် standard input သို့မဟုတ် standard output အဖြစ် သတ်မှတ်ရန်။
- `--` : Special argument ဖြစ်ပြီး ၎င်းနောက်မှ လာသော `-` ပါသော အရာများကို Flag အဖြစ် မ သတ်မှတ်တော့ဘဲ မူလ Argument အဖြစ် သတ်မှတ်ပေးရန် (ဥပမာ `rm -- -r`)။

Window managers

“floating” window manager များအပြင် အခြား အမျိုးအစားတစ်ခုမှာ “tiling” window manager ဖြစ်ပါသည်။ Tiling window manager တွင် Window များသည် တစ်ခုပေါ်တစ်ခု ထပ်မနေဘဲ Screen ပေါ်တွင် Tiles များအဖြစ် စီတန်းနေမည် ဖြစ်သည်။

VPNs

VPN သည် သင်၏ ISP အစား VPN Provider ထံသို့ ယုံကြည်မှုကို ပြောင်းလဲလိုက်ခြင်း ဖြစ်ပါသည်။ အကျိုးကျေးဇူး ရှိမရှိ သေချာစွာ စိစစ်ပါ။ အသုံးများသော WireGuard အတွက် **WireGuard** (https://www.wireguard.com/) ကို ကြည့်ပါ၊ MIT ကျောင်းသားများအတွက် **MIT VPN** (https://ist.mit.edu/vpn) ရှိသည်။

Markdown

Markdown (https://commonmark.org/help/) သည် ရိုးရှင်းသော စာသားများ ရေးသားရန်အတွက် Lightweight markup language ဖြစ်ပါသည်။ - `*italic*` - `**bold**` - `# Heading` - `- list item` - `` `code` `` - `[name](url)`

ဤသင်ခန်းစာ Document များ အားလုံးကို Markdown ဖြင့် ရေးသားထားခြင်း ဖြစ်သည်။

Hammerspoon (desktop automation on macOS)

Hammerspoon (https://www.hammerspoon.org/) သည် macOS အတွက် Desktop automation framework ဖြစ်ပါသည်။ Lua script များ ရေးသား၍ စနစ်ကို စီမံနိုင်ပါသည်။

Booting + Live USBs

BIOS (https://en.wikipedia.org/wiki/BIOS)/**UEFI** (https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface) သည် ကွန်ပျူတာ စတင်ချိန်တွင် စနစ်ကို စတင်ပေးသည်။ Boot Menu မှတဆင့် Alternate device မှ Boot တက်နိုင်သည်။

Live USBs (https://en.wikipedia.org/wiki/Live_USB) တွင် Operating system ပါရှိပြီး၊ **UNetbootin** (https://unetbootin.github.io/) ကဲ့သို့သော Tool ဖြင့် ဖန်တီးနိုင်သည်။

Docker, Vagrant, VMs, Cloud, OpenStack

Virtual machines (https://en.wikipedia.org/wiki/Virtual_machine) နှင့် Containers များသည် စနစ်တစ်ခုလုံးကို အတုယူ ဖန်တီးပေးသည်။ **Vagrant** (https://www.vagrantup.com/)၊ **Docker** (https://www.docker.com/)၊ Cloud Provider များ (**AWS** (https://aws.amazon.com/), **Google Cloud** (https://cloud.google.com/), **Azure** (https://azure.microsoft.com/), **DigitalOcean** (https://www.digitalocean.com/))။

MIT CSAIL ကျောင်းသားများအတွက် **CSAIL OpenStack** (https://tig.csail.mit.edu/shared-computing/open-stack/) တွင် အခမဲ့ သုံးနိုင်သည်။

Notebook programming

Jupyter (https://jupyter.org/) (Python) သို့မဟုတ် **Wolfram Mathematica** (https://www.wolfram.com/mathematica/) တို့သည် Notebook programming အတွက် အသုံးများပါသည်။

GitHub

GitHub (https://github.com/) တွင် Open-source ပရောဂျက်များစွာ ရှိသည်။ - Issue တင်ခြင်း - Pull request တင်ခြင်း

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. karabiner-elements	https://karabiner-elements.pqrs.org/	
2. skhd	https://github.com/koekeishiya/skhd	
3. BetterTouchTool	https://folivora.ai/	
4. xmodmap	https://wiki.archlinux.org/index.php/Xmodmap	
5. Autokey	https://github.com/autokey/autokey	
6. AutoHotkey	https://www.autohotkey.com/	
7. SharpKeys	https://www.randyrants.com/category/sharpkeys/	
8. QMK	https://docs.qmk.fm/	
9. `cron`	https://www.man7.org/linux/man-pages/man8/cron.8.html	
10. FUSE	https://en.wikipedia.org/wiki/Filesystem_in_Userspace	
11. sshfs	https://github.com/libfuse/sshfs	
12. rclone	https://rclone.org/commands/rclone_mount/	
13. gocryptfs	https://nuetzlich.net/gocryptfs/	
14. kbfs	https://keybase.io/docs/kbfs	
15. borgbackup	https://borgbackup.readthedocs.io/en/stable/usage/mount.html	

Resource / Description	URL	Micro QR
16. Backups သင်ခန်းစာ	https://missing-semester-my.github.io/2019/backups	
17. `jq`	https://stedolan.github.io/jq/	
18. OAuth	https://www.oauth.com/	
19. IFTTT	https://ifttt.com/	
20. WireGuard	https://www.wireguard.com/	
21. MIT VPN	https://ist.mit.edu/vpn	
22. Markdown	https://commonmark.org/help/	
23. Hammerspoon	https://www.hammerspoon.org/	
24. BIOS	https://en.wikipedia.org/wiki/BIOS	
25. UEFI	https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface	
26. Live USBs	https://en.wikipedia.org/wiki/Live_USB	
27. UNetbootin	https://unetbootin.github.io/	
28. Virtual machines	https://en.wikipedia.org/wiki/Virtual_machine	
29. Vagrant	https://www.vagrantup.com/	
30. Docker	https://www.docker.com/	
31. AWS	https://aws.amazon.com/	

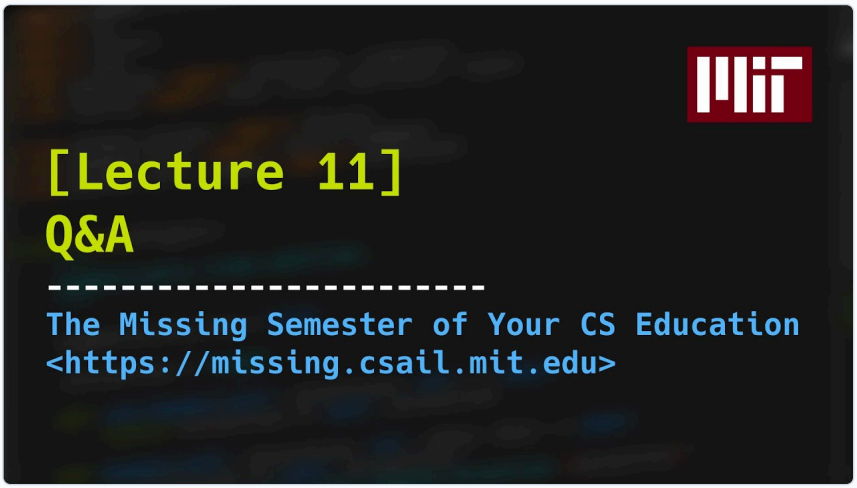
Resource / Description	URL	Micro QR
32. Google Cloud	https://cloud.google.com/	
33. Azure	https://azure.microsoft.com/	
34. DigitalOcean	https://www.digitalocean.com/	
35. CSAIL OpenStack	https://tig.csail.mit.edu/shared-computing/open-stack/	
36. Jupyter	https://jupyter.org/	
37. Wolfram Mathematica	https://www.wolfram.com/mathematica/	
38. GitHub	https://github.com/	

Q&A

အနှစ်ချုပ် - Operating systems, shell scripting, tool အကြံပြုချက်များ နှင့် အခြား ခေါင်းစဉ်များ အပေါ် ကျောင်းသားများ၏ မေးခွန်းများအတွက် အဖြေများ။

▶ LECTURE VIDEO REFERENCE

Q&A



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=Wz50FvGG6xU

နောက်ဆုံး သင်ခန်းစာအဖြစ် ကျောင်းသားများ ပေးပို့ထားသော မေးခွန်းများကို ဖြေကြားခဲ့ပါသည် -

- Any recommendations on learning Operating Systems related topics like processes, virtual memory, interrupts, memory management, etc
- What are some of the tools you'd prioritize learning first?
- When do I use Python versus a Bash scripts versus some other language?
- What is the difference between `source script.sh` and `./script.sh`
- What are the places where various packages and tools are stored and how does referencing them work? What even is `/bin` or `/lib` ?
- Should I `apt-get install` a python-whatever, or `pip install` whatever package?
- What's the easiest and best profiling tools to use to improve performance of my code?
- What browser plugins do you use?
- What are other useful data wrangling tools?
- What is the difference between Docker and a Virtual Machine?
- What are the advantages and disadvantages of each OS and how can we choose between them (e.g. choosing the best Linux distribution for our purposes)?
- Vim vs Emacs?
- Any tips or tricks for Machine Learning applications?
- Any more Vim tips?
- What is 2FA and why should I use it?
- Any comments on differences between web browsers?

Any recommendations on learning Operating Systems related topics like processes, virtual memory, interrupts, memory management, etc

ပထမဦးစွာ ဤခေါင်းစဉ်များသည် အလွန် Low-level ကျသောကြောင့် အားလုံးကို ကျွမ်းကျင်ရန် လိုမလိုမှာ မသေချာပါ။ Kernel ကို ရေးသားခြင်း သို့မဟုတ် ပြင်ဆင်ခြင်း ကဲ့သို့သော Low-level code များ ရေးသားချိန်တွင်မှ သက်ရောက်မည် ဖြစ်သည်။

သင်ယူရန် ကောင်းမွန်သော သင်ခန်းစာများ - - MIT ၏ 6.828 အတန်း (https://pdos.csail.mit.edu/6.828/) - Graduate level Operating System Engineering အတန်း ဖြစ်သည်။ - Modern Operating Systems (4th ed) - Andrew S. Tanenbaum ရေးသားသည်။ - The Design and Implementation of the FreeBSD Operating System - Writing an OS in Rust (https://os.phil-opp.com/)

What are some of the tools you'd prioritize learning first?

ဦးစားပေးသင့်သော ခေါင်းစဉ်အချို့ - - Keyboard ကို ပိုမို အသုံးပြုပြီး Mouse ကို နည်းပါးစွာ သုံးတတ်အောင် လေ့ကျင့်ပါ။ - မိမိ အသုံးပြုသော Editor ကို ကျွမ်းကျင်စွာ သုံးတတ်အောင် လေ့ကျင့်ပါ။ - တူညီသော အလုပ်များကို အလိုအလျောက် ပြုလုပ်နိုင်ရန် လေ့လာပါ...။ - Git ကဲ့သို့သော Version control tool များကို လေ့လာပါ။

When do I use Python versus a Bash scripts versus some other language?

ယေဘုယျအားဖြင့် Bash script သည် Command အနည်းငယ် runမည့် ရိုးရှင်းသော အလုပ်များအတွက် အသုံးဝင်ပါသည်။ ကြီးမားသော ပရိုဂရမ်များအတွက် Bash တွင် အားနည်းချက်များ ရှိပါသည် - - ကွက်လပ်များ (spaces) ပါပါက Bug ဖြစ်ပေါ်နိုင်ခြင်း။ - Code များကို ပြန်လည် အသုံးပြုရန် ခက်ခဲခြင်း (Library အယူအဆ မရှိခြင်း)။ - `$?` သို့မဟုတ် `$@` ကဲ့သို့သော Magic string များကို သုံးရခြင်း။

ထို့ကြောင့် ကြီးမားသော Script များအတွက် Python သို့မဟုတ် Ruby ကို အကြံပြုပါသည်။

What is the difference between `source script.sh` and `./script.sh`

`source` သည် လက်ရှိ Bash session တွင် Command များကို runပေးသဖြင့် Directory ပြောင်းခြင်း၊ Function သတ်မှတ်ခြင်းတို့သည် လက်ရှိ Session တွင် ကျန်ရစ်မည် ဖြစ်သည်။ `./script.sh` မူ Bash session အသစ်တစ်ခု ဖွင့်၍ runသဖြင့် Session ပြီးပါက မူလနေရာသို့ ပြန်ရောက်မည် ဖြစ်သည်။

What are the places where various packages and tools are stored and how does referencing them work? What even is `/bin` or `/lib`?

- `/bin` - Essential command binaries
- `/sbin` - Root မှ runမည့် Essential system binaries
- `/dev` - Device files
- `/etc` - System-wide configuration files
- `/home` - User များ၏ Home directories
- `/lib` - System program များအတွက် Common libraries
- `/opt` - Optional application software

- `/sys` - System configuration (ပထမ သင်ခန်းစာတွင် ပါဝင်ခဲ့သည်)
- `/tmp` - Temporary files (Reboot လုပ်ပါက ပျက်စီးမည်)
- `/usr/` - Read only user data
  - `/usr/bin` - Non-essential command binaries
  - `/usr/sbin` - Non-essential system binaries
  - `/usr/local/bin` - User compiled binaries
- `/var` - Variable files (logs, caches)

Should I `apt-get install` a python-whatever, or `pip install` whatever package?

ပရိုဂရမ်မင်း ဘာသာစကား သီးသန့် Package manager (ဥပမာ `pip`) ကို အသုံးပြုရန်နှင့် မူလ စနစ်ကို မထိခိုက်စေရန် Virtual environment (`virtualenv`) များကို အသုံးပြုရန် အကြံပြုပါသည်။

What's the easiest and best profiling tools to use to improve performance of my code?

အလွယ်ကူဆုံးမှာ **print timing** (https://missing-semester-my.github.io/2020/debugging-profiling/#timing) ဖြစ်ပါသည်။ အဆင့်မြင့် Tool များအတွက် Valgrind ၏ **Callgrind** (https://valgrind.org/docs/manual/cl-manual.html)၊ `perf` (https://www.brendangregg.com/perf.html) နှင့် **Flamegraphs** (https://www.brendangregg.com/flamegraphs.html) တို့ကို အသုံးပြုနိုင်ပါသည်။

What browser plugins do you use?

- **uBlock Origin** (https://github.com/gorhill/uBlock) - Ad blocker
- **Stylus** (https://github.com/openstyles/stylus/) - Custom CSS
- Full Page Screen Capture
- **Multi Account Containers** (https://addons.mozilla.org/en-US/firefox/addon/multi-account-containers/)
- Password Manager Integration
- **Vimium** (https://github.com/philc/vimium)

What are other useful data wrangling tools?

`jq` , `pup` , Perl, `column -t` တို့ ဖြစ်ကြပါသည်။ Python တွင် **pandas** (https://pandas.pydata.org/) library သည် CSV များနှင့် အခြား Tabular data များအတွက် အလွန် ကောင်းမွန်ပါသည်။

What is the difference between Docker and a Virtual Machine?

Virtual machine သည် Kernel အပါအဝင် OS တစ်ခုလုံးကို runပေးသည်။ Docker (Containers) ကမူ Host စနစ်၏ Kernel ကို မျှဝေ သုံးစွဲသဖြင့် Overhead နည်းပါးသည်။

What are the advantages and disadvantages of each OS and how can we choose between them (e.g. choosing the best Linux distribution for our purposes)?

Linux distro အများစုသည် လုပ်ဆောင်ချက် ဆင်တူကြသည်။ အဓိက ကွာခြားချက်မှာ Package update ပေါ်မူတည်သည် (Arch Linux တွင် Rolling update ဖြစ်ပြီး၊ Debian/Ubuntu တွင် Conservative ဖြစ်သည်)။ Desktop နှင့် Server အတွက် Debian သို့မဟုတ် Ubuntu ကို အကြိုပြုပါသည်။

Vim vs Emacs?

ဆရာများ အားလုံး Vim ကို အဓိက သုံးကြသော်လည်း Emacs သည်လည်း ကောင်းမွန်သော Alternative ဖြစ်သည်။

Any tips or tricks for Machine Learning applications?

ML တွင် စမ်းသပ်မှုများကို ခြေရာခံရန် JSON file များ၊ Shell tool များနှင့် Automation များကို သုံးနိုင်သည်။

Any more Vim tips?

- Plugins (VimAwesome)
- Marks (`m<X>` နှင့် `'<X>`)
- Navigation (`Ctrl+O` နှင့် `Ctrl+I`)
- Undo Tree (**undotree** (https://github.com/mbbill/undotree))

- Persistent undo (`undofile` နှင့် `undodir`)
- Leader Key

What is 2FA and why should I use it?

2FA သည် စကားဝှက်အပြင် Hardware device ကို သုံးစေ၍ လုံခြုံရေး အလွှာတစ်ခု ထပ်မံ ပေါင်းထည့်ပေးခြင်း ဖြစ်သည်။ **YubiKey** (https://www.yubico.com/) (U2F) ကို အကြံပြုပါသည်။

Any comments on differences between web browsers?

2FA နှင့် လုံခြုံရေး အပြင် Firefox ကို အဓိက အကြံပြုပါသည်။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

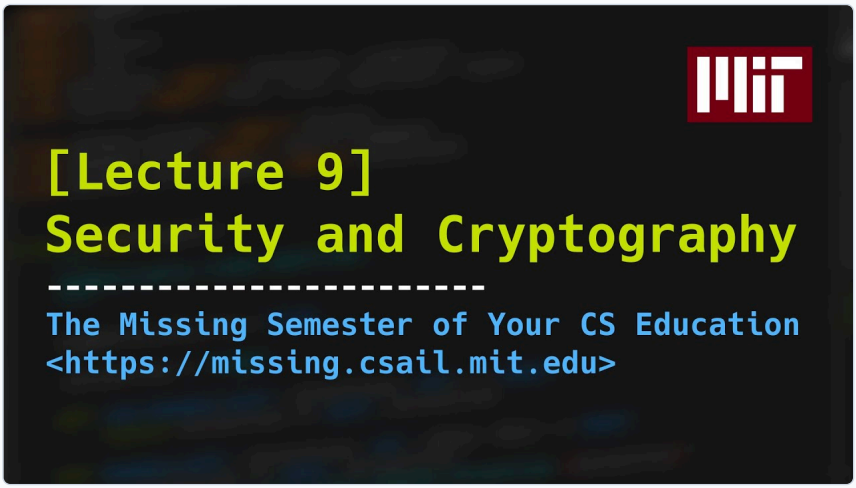
Resource / Description	URL	Micro QR
1. MIT ၏ 6.828 အတန်း	https://pdos.csail.mit.edu/6.828/	
2. Writing an OS in Rust	https://os.phil-opp.com/	
3. print timing	https://missing-semester-my.github.io/2020/debugging-profiling/#timing	
4. Callgrind	https://valgrind.org/docs/manual/cl-manual.html	
5. `perf`	https://www.brendangregg.com/perf.html	
6. Flamegraphs	https://www.brendangregg.com/flamegraphs.html	
7. uBlock Origin	https://github.com/gorhill/uBlock	
8. Stylus	https://github.com/openstyles/stylus/	
9. Multi Account Containers	https://addons.mozilla.org/en-US/firefox/addon/multi-account-containers/	
10. Vimium	https://github.com/philc/vimium	
11. pandas	https://pandas.pydata.org/	
12. undotree	https://github.com/mbbill/undotree	
13. YubiKey	https://www.yubico.com/	

Security and Cryptography

အနှစ်ချုပ် - Hashes နှင့် key derivation functions ကဲ့သို့သော Cryptographic primitives များနှင့် Git၊ SSH ကဲ့သို့သော Tool များက ယင်းတို့ကို မည်သို့ သုံးစွဲပုံများ လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Security and Cryptography



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=tjwobAmnKTo

လွန်ခဲ့သော နှစ်၏ [လုံခြုံရေး နှင့် သီးသန့်လုံခြုံမှု သင်ခန်းစာ](https://missing-semester-my.github.io/2019/security/) သည် ကွန်ပျူတာ *အသုံးပြုသူ* တစ်ယောက်အနေဖြင့် ပိုမို လုံခြုံအောင် မည်သို့ နေထိုင်ရမည်ကို အဓိကထားခဲ့သည်။ ဤနှစ်တွင်မူ Git ရှိ Hash function များ၊ SSH ရှိ Key derivation function များနှင့် Symmetric/Asymmetric cryptosystems များကဲ့သို့ ဤအတန်းတွင် ယခင်က သင်ကြားခဲ့သော Tool များကို နားလည်စေမည့် လုံခြုံရေး နှင့် Cryptography အယူအဆများကို အဓိကထား သင်ကြားပါမည်။

ဤသင်ခန်းစာသည် ကွန်ပျူတာစနစ် လုံခြုံရေး (**6.858** (https://css.csail.mit.edu/6.858/)) သို့မဟုတ် Cryptography (**6.857** (https://courses.csail.mit.edu/6.857/) နှင့် 6.875) အတန်းအပြည့်အစုံအတွက် အစားထိုးခြင်း မဟုတ်ပါ။ လုံခြုံရေးနှင့် ပတ်သက်သော စနစ်တကျ သင်တန်း မတက်ရောက်ဘဲ လုံခြုံရေး အလုပ်များကို မလုပ်ဆောင်ပါနှင့်။ ကျွမ်းကျင်သူ မဟုတ်ပါက မိမိကိုယ်တိုင် **Crypto ရေးသားခြင်း မပြုပါနှင့်** (https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html)။ ထို စည်းမျဉ်းသည် စနစ် လုံခြုံရေးအတွက်လည်း အတူတူပင် ဖြစ်ပါသည်။

ဤသင်ခန်းစာသည် အခြေခံ Cryptography အယူအဆများကို လွတ်လပ်စွာ (သို့သော် လက်တွေ့ကျကျ) သင်ကြားထားခြင်း ဖြစ်သည်။ ဤသင်ခန်းစာသည် လုံခြုံသော စနစ်များ သို့မဟုတ် Cryptographic protocol များကို မည်သို့ *ဒီဇိုင်းထုတ်ရမည်* ကို သင်ကြားရန် မလုံလောက်သော်လည်း သင် အသုံးပြုနေပြီးသား ပရိုဂရမ်များနှင့် Protocol များကို ယေဘုယျ နားလည်စေရန် ကူညီပေးလိမ့်မည်ဟု မျှော်လင့်ပါသည်။

Entropy

Entropy (https://en.wikipedia.org/wiki/Entropy_(information_theory)) ဆိုသည်မှာ ရန်ဒမ်ဖြစ်မှု (randomness) ကို တိုင်းတာခြင်း ဖြစ်သည်။ ဤသည်မှာ ဥပမာ စကားဝှက် တစ်ခု၏ ကြံ့ခိုင်မှုကို ဆုံးဖြတ်ရာတွင် အသုံးဝင်ပါသည်။

!**XKCD 936: Password Strength** (https://imgs.xkcd.com/comics/password_strength.png)

အထက်ပါ **XKCD comic** (https://xkcd.com/936/) က သရုပ်ဖော်ထားသည့်အတိုင်း “correcthorsebatterystaple” ကဲ့သို့သော စကားဝှက်သည် “Tr0ub4dor&3” ထက် ပိုမို လုံခြုံမှု ရှိပါသည်။ သို့သော် ဤကဲ့သို့သော အရာကို မည်သို့ ပမာဏ သတ်မှတ်မနည်း။

Entropy ကို *bits* ဖြင့် တိုင်းတာပြီး၊ ဖြစ်နိုင်ခြေရှိသော အစုအဝေးမှ ညီမျှစွာ သီးသန့် ရွေးချယ်သည့်အခါ Entropy သည် `log_2(# of possibilities)` နှင့် တူညီပါသည်။ မျှတသော အကြွေစေ့ တစ်ပြား ပစ်လိုက်ခြင်းသည် Entropy 1 bit ရရှိပါသည်။ ၆ မျက်နှာပါ အန်စာတုံး တစ်ခု ပစ်လိုက်ခြင်းသည် Entropy ~2.58 bits ရရှိပါသည်။

တိုက်ခိုက်သူသည် စကားဝှက်၏ *ပုံစံ (model)* ကို သိရှိသော်လည်း စကားဝှက် ရွေးချယ်ရန် အသုံးပြုထားသော Random ဖြစ်မှုကို (အန်စာတုံး ပစ်ခြင်း (https://en.wikipedia.org/wiki/Diceware) စသည်) မသိရှိပါ ဟု သတ်မှတ်ရပါမည်။

Entropy မည်မျှ လုံလောက်သနည်း။ ခြိမ်းခြောက်မှု မူဘောင်ပေါ်မူတည်ပါသည်။ အွန်လိုင်းမှ ခန့်မှန်းခြင်း အတွက် XKCD comic တွင် ထောက်ပြထားသကဲ့သို့ ~40 bits ၏ Entropy သည် အတော်အတန် ကောင်းမွန်ပါသည်။ အော့ဖ်လိုင်းမှ ခန့်မှန်းခြင်းကို ခံနိုင်ရည်ရှိစေရန် ပိုမို ခိုင်မာသော စကားဝှက် လိုအပ်ပါမည် (ဥပမာ 80 bits သို့မဟုတ် ထိုထက်ပို၍)။

Hash functions

cryptographic hash function (https://en.wikipedia.org/wiki/Cryptographic_hash_function) သည် မည်သည့် ဆိုဒ်ရှိမဆို အချက်အလက်များကို ပုံသေ ဆိုဒ်တစ်ခုသို့ ပြောင်းလဲပေးပြီး အထူး ဂုဏ်သတ္တိများ ရှိကြသည်။ Hash function ၏ ယေဘုယျ ပုံစံမှာ အောက်ပါအတိုင်း ဖြစ်ပါသည် -

```
hash(value: array<byte>) -> vector<byte, N>  (for some fixed N)
```

Hash function ၏ ဥပမာတစ်ခုမှာ Git တွင် အသုံးပြုထားသော **SHA1** (https://en.wikipedia.org/wiki/SHA-1) ဖြစ်ပါသည်။ ၎င်းသည် မည်သည့် ဆိုဒ်ရှိမဆို Input များကို 160-bit Output (Hexadecimal စာလုံး ၄၀) အဖြစ် ပြောင်းလဲပေးသည်။ `sha1sum` command ဖြင့် Input အပေါ် SHA1 hash ကို စမ်းသပ်ကြည့်နိုင်ပါသည် -

```
$ printf 'hello' | sha1sum
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
$ printf 'hello' | sha1sum
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
$ printf 'Hello' | sha1sum
f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
```

အဆင့်မြင့် ရူထောင့်မှကြည့်လျှင် Hash function ကို ပြန်လည် ပြောင်းလဲရန် ခက်ခဲသော Random ကဲ့သို့ (သေချာသော) Function ဟု စဉ်းစားနိုင်ပါသည်။ Hash function တွင် အောက်ပါ ဂုဏ်သတ္တိများ ရှိပါသည် -

- Deterministic: တူညီသော Input သည် အမြဲတမ်း တူညီသော Output ကို ထုတ်ပေးသည်။
- Non-invertible: သီးခြား Output `h` အတွက် `hash(m) = h` ဖြစ်မည့် Input `m` ကို ရှာဖွေရန် ခက်ခဲသည်။
- Target collision resistant: Input `m_1` ဖြင့် `hash(m_1) = hash(m_2)` ဖြစ်မည့် ကွဲပြားသော Input `m_2` ကို ရှာဖွေရန် ခက်ခဲသည်။
- Collision resistant: `hash(m_1) = hash(m_2)` ဖြစ်မည့် Input နှစ်ခု `m_1` နှင့် `m_2` ကို ရှာဖွေရန် ခက်ခဲသည်။

သတိပြုရန်မှာ SHA-1 ကို ပြင်းထန်သော Cryptographic hash function တစ်ခုအဖြစ် **မသတ်မှတ်တော့ပါ** (https://web.archive.org/web/20260207211148/https://shattered.io/)။ သီးခြား Hash function များကို အကြံပြုခြင်း

သည် ဤသင်ခန်းစာ၏ နယ်ပယ်ထက် ကျော်လွန်ပါသည်။ လုံခြုံရေး အလုပ်များ လုပ်ဆောင်ပါက formal training ရယူရန် လိုအပ်ပါသည်။

Applications

- Content-addressed storage အတွက် Git။ **hash function** (https://en.wikipedia.org/wiki/Hash_function) ၏ အယူအဆသည် ပိုမို ယေဘုယျ ကျသည်။ Git က Cryptographic hash function ကို အဘယ်ကြောင့် သုံးသနည်း။
- ဖိုင်တစ်ခု၏ အကြောင်းအရာ အတိုချုပ်။ စော့ဖ်ဝဲလ်များကို Mirror များမှ ဒေါင်းလုဒ် ရယူလေ့ ရှိကြသဖြင့် တရားဝင် ဝဘ်ဆိုက်များက ဒေါင်းလုဒ် ဖိုင်နှင့်အတူ Hash ကို တင်ပေးထားလေ့ ရှိသည်။
- **Commitment schemes** (https://en.wikipedia.org/wiki/Commitment_scheme)။ သီးခြား တန်ဖိုးတစ်ခုကို သတ်မှတ်ထားပြီး နောက်မှ ထုတ်ပြလိုသည့် အခါမျိုး ဖြစ်သည်။ ဥပမာ အကြွေစေ့ ပစ်ခြင်း။ `r = random()` ကို ရွေးပြီး `h = sha256(r)` ကို မျှဝေသည်။ အခြားသူက ခန်းမှန်းပြီးနောက် `r` ကို ထုတ်ပြ၍ စိစစ်စေသည်။

Key derivation functions

Cryptographic hashes များနှင့် ဆက်စပ်နေသော အယူအဆတစ်ခုမှာ **key derivation functions** (https://en.wikipedia.org/wiki/Key_derivation_function) (KDFs) ဖြစ်ပြီး အခြား Cryptographic algorithm များတွင် Key အဖြစ် သုံးရန် ပုံသေ အလျားရှိ Output ထုတ်ပေးခြင်း အပါအဝင် အသုံးချမှုများစွာတွင် သုံးသည်။ သာမန်အားဖြင့် အော့ဖ်လိုင်း တိုက်ခိုက်မှုများကို နှောင့်နှေးစေရန် KDF များကို တမင် နှေးကွေးအောင် ပြုလုပ်ထားသည်။

Applications

- Passphrases မှတဆင့် အခြား Cryptographic algorithm များတွင် သုံးမည့် Key များ ထုတ်လုပ်ခြင်း။
- လော့ဂ်အင် အချက်အလက်များ သိမ်းဆည်းခြင်း။ Plaintext password များကို သိမ်းခြင်းမှာ မကောင်းပါ; User တိုင်းအတွက် Random **salt** (https://en.wikipedia.org/wiki/Salt_(cryptography)) `salt = random()` ကို ထုတ်ယူ၍ `KDF(password + salt)` ကို သိမ်းဆည်းရန် ဖြစ်သည်။

Symmetric cryptography

အကြောင်းအရာများကို ဖုံးကွယ်ခြင်းသည် Cryptography အကြောင်း စဉ်းစားလျှင် ပထမဆုံး တွေးမိသော အရာ ဖြစ်သည်။ Symmetric cryptography သည် အောက်ပါ လုပ်ဆောင်ချက်များဖြင့် ပြုလုပ်သည် -

```
keygen() -> key  (this function is randomized)

encrypt(plaintext: array<byte>, key) -> array<byte>  (the ciphertext)
decrypt(ciphertext: array<byte>, key) -> array<byte>  (the plaintext)
```

Encrypt function သည် Output (ciphertext) မှ Key မရှိဘဲ Input (plaintext) ကို ရှာဖွေရန် ခက်ခဲသော ဂုဏ်သတ္တိ ရှိသည်။ Decrypt function မူ `decrypt(encrypt(m, k), k) = m` ဖြစ်သည်။

ယနေ့ခေတ် အသုံးများသော Symmetric cryptosystem ၏ ဥပမာမှာ **AES** (https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) ဖြစ်သည်။

Applications

- မယုံကြည်ရသော Cloud service တွင် သိမ်းရန် ဖိုင်များကို Encrypt လုပ်ခြင်း။ KDF များဖြင့် တွဲဖက်၍ Passphrase ဖြင့် Encrypt လုပ်နိုင်ပါသည်။ `key = KDF(passphrase)` ထုတ်ယူ၍ `encrypt(file, key)` ကို သိမ်းသည်။

Asymmetric cryptography

“Asymmetric” ဟု ခေါ်ဆိုရခြင်းမှာ တာဝန်ကွဲပြားသော Key နှစ်ခု ပါဝင်သောကြောင့် ဖြစ်သည်။ Private key သည် လျှို့ဝှက် ထားရမည် ဖြစ်ပြီး Public key မူ အများသို့ မျှဝေနိုင်ပြီး လုံခြုံရေးကို မထိခိုက်ပါ။ အောက်ပါ လုပ်ဆောင်ချက်များ ပါဝင်ပါသည် -

```
keygen() -> (public key, private key)  (this function is randomized)

encrypt(plaintext: array<byte>, public key) -> array<byte>  (the ciphertext)
decrypt(ciphertext: array<byte>, private key) -> array<byte>  (the plaintext)

sign(message: array<byte>, private key) -> array<byte>  (the signature)
verify(message: array<byte>, signature: array<byte>, public key) -> bool  (whet
her or not the signature is valid)
```

Message ကို *public* key ဖြင့် Encrypt လုပ်နိုင်သည်၊ *private* key မရှိဘဲ Plaintext ကို ရရှိရန် ခက်ခဲသည်။ `decrypt(encrypt(m, public key), private key) = m` ဖြစ်သည်။

Symmetric နှင့် Asymmetric ကို သော့ခလောက်များဖြင့် ယှဉ်တွဲနိုင်ပါသည်။ Symmetric သည် တံခါး သော့ခလောက်ကဲ့သို့ သော့ရှိသူတိုင်း ပိတ်/ဖွင့် နိုင်သည်။ Asymmetric မူ သော့ပါသော သော့ခလောက်ကဲ့သို့ အများသို့ သော့ခလောက် (Public key) ပေးထားပြီး သော့ (Private key) ကို မိမိတစ်ဦးတည်း ကိုင်ဆောင်ထားခြင်း ဖြစ်သည်။

Sign/verify function များသည် လက်မှတ် အတု ပြုလုပ်ရန် ခက်ခဲသော ဂုဏ်သတ္တိ ရှိသည်။ Private key မရှိဘဲ `verify(message, signature, public key)` ကို true ထွက်အောင် Signature ထုတ်ရန် ခက်ခဲသည်။ `verify(message, sign(message, private key), public key) = true` ဖြစ်သည်။

Applications

- **PGP email encryption** (https://en.wikipedia.org/wiki/Pretty_Good_Privacy)။
- သီးသန့် မက်ဆေ့ဂျ် ပို့ခြင်း (**Signal** (https://signal.org/) နှင့် **Keybase** (https://keybase.io/))။
- စော့ဖ်ဝဲလ် လက်မှတ်ရေးထိုးခြင်း (Git တွင် GPG-signed commits များ သုံးနိုင်သည်)။

Key distribution

Asymmetric-key cryptography သည် အလွန် ကောင်းမွန်သော်လည်း Public key များကို လူအမှန်နှင့် တွဲဖက်ဖြန့်ဝေရန် စိန်ခေါ်မှု ရှိသည်။ Signal တွင် ပထမဆုံး အသုံးပြုမှုကို ယုံကြည်ခြင်း သို့မဟုတ် ပြင်ပမှ စိစစ်ခြင်း သုံးသည်။ PGP တွင် **web of trust** (https://en.wikipedia.org/wiki/Web_of_trust) သုံးသည်။ Keybase တွင် **social proof** (https://keybase.io/blog/chat-apps-softer-than-tofu) သုံးသည်။

Case studies

Password managers

လူတိုင်း အသုံးပြုသင့်သော Tool ဖြစ်သည် (ဥပမာ KeePassXC (https://keepassxc.org/), pass (https://git.zx2c4.com/password-store/about/), 1Password (https://1password.com))။ မတူညီသော စကားဝှက်များကို အသုံးပြုစေနိုင်ပြီး၊ KDF မှတဆင့် Passphrase ဖြင့် Symmetric cipher ဖြင့် ဖိုင်အဖြစ် သိမ်းဆည်းပေးသည်။

Two-factor authentication

Two-factor authentication (https://en.wikipedia.org/wiki/Multi-factor_authentication) (2FA) သည် စကားဝှက် (“သိထားသော အရာ”) နှင့်အတူ Authenticator (“ပိုင်ဆိုင်သော အရာ”) ကို တွဲဖက် အသုံးပြုစေခြင်း ဖြစ်သည်။

Full disk encryption

ကွန်ပျူတာ အခိုးခံရပါက Data များကို ကာကွယ်ရန် Full disk encryption ကို သုံးပါ။ Linux တွင် cryptsetup + LUKS (https://wiki.archlinux.org/index.php/Dm-crypt/Encrypting_a_non-root_file_system)၊ Windows တွင် BitLocker (https://fossbytes.com/enable-full-disk-encryption-windows-10/)၊ macOS တွင် FileVault (https://support.apple.com/en-us/HT204837)။

Private messaging

Signal (https://signal.org/) သို့မဟုတ် Keybase (https://keybase.io/) ကို သုံးပါ။ End-to-end security ကို Asymmetric key ဖြင့် စတင်တည်ဆောက်ထားသည်။

SSH

`ssh-keygen` runသည့်အခါ Asymmetric key pair (`public_key, private_key`) ထုတ်ပေးသည်။ `ssh-keygen` က Passphrase တောင်းပြီး KDF မှတဆင့် Key ထုတ်ယူကာ Private key ကို Disc ပေါ်တွင် Encrypt လုပ်၍ သိမ်းသည်။

Server တွင် Client ၏ Public key ရှိသည့်အခါ (`.ssh/authorized_keys` တွင်)၊ Client သည် Challenge-response မှတဆင့် Asymmetric signature ဖြင့် မိမိ ပိုင်ဆိုင်ကြောင်း သက်သေပြ၍ လော့ဂ်အင် ဝင်ရောက်သည်။

Resources

- **Last year's notes** (https://missing-semester-my.github.io/2019/security/)
- **Cryptographic Right Answers** (https://latacora.micro.blog/2018/04/03/cryptographic-right-answers.html)

Exercises

1. **Entropy.**

```
1. စကားဝှက်တစ်ခုကို စာလုံး ၁၀၀,၀၀၀ ပါသော အဘိဓာန်မှ သီးခြား စကားလုံး ၄ ခု တွဲဖက်၍ ရွေးချယ်သည်ဆိုပါစို့ (ဥပမာ `correcthorsebatterystaple`)။ Entropy bits မည်မျှ ရှိသနည်း။
2. စကားဝှက်တစ်ခုကို စာလုံး ၈ လုံးပါသော Random အက္ခရာ/ကိန်းဂဏန်းများဖြင့် ရွေးချယ်သည်ဆိုပါစို့ (ဥပမာ `rg8Ql34g`)။ Entropy bits မည်မျှ ရှိသနည်း။
3. မည်သည့် စကားဝှက်က ပိုမို ခိုင်မာသနည်း။
4. တိုက်ခိုက်သူက ၁ စက္ကန့်လျှင် စကားဝှက် ၁၀,၀၀၀ ခန့်မှန်းနိုင်ပါက၊ စကားဝှက်တစ်ခုစီကို ခန့်မှန်းမိရန် ပျမ်းမျှ အချိန်မည်မျှ ကြာမည်နည်း။
```

2. **Cryptographic hash functions.** Debian image တစ်ခုကို **Mirror** (https://www.debian.org/CD/http-ftp/) မှ ဒေါင်းလုဒ် လုပ်ပြီး Hash စိစစ်ပါ။

3. **Symmetric cryptography.** OpenSSL ဖြင့် AES encryption သုံး၍ ဖိုင်ကို Encrypt လုပ်ပါ: `openssl aes-256-cbc -salt -in {input filename} -out {output filename}` ။ `cat` သို့မဟုတ် `hexdump` ဖြင့် ကြည့်ပါ။ `openssl aes-256-cbc -d -in {input filename} -out {output filename}` ဖြင့် Decrypt ပြန်လုပ်ပြီး `cmp` ဖြင့် တူမတူ စစ်ပါ။

4. **Asymmetric cryptography.**

```
1. SSH keys များ တပ်ဆင်ပါ။
2. GPG တပ်ဆင်ပါ။
3. Encrypted email ပို့ပါ။
4. `git commit -S` ဖြင့် Git commit ကို Sign လုပ်ပါ။ `git show --show-signature` ဖြင့် စိစစ်ပါ။
```

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. လုံခြုံရေး နှင့် သီးသန့်လုံခြုံမှု သင်ခန်းစာ	https://missing-semester-my.github.io/2019/security/	
2. 6.858	https://css.csail.mit.edu/6.858/	
3. 6.857	https://courses.csail.mit.edu/6.857/	
4. Crypto ရေးသားခြင်း မပြုပါ နှင့်	https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html	
5. Entropy	https://en.wikipedia.org/wiki/Entropy_(information_theory	
6. XKCD 936: Password Strength	https://imgs.xkcd.com/comics/password_strength.png	
7. XKCD comic	https://xkcd.com/936/	
8. အန်စာတုံး ပစ်ခြင်း	https://en.wikipedia.org/wiki/Diceware	
9. cryptographic hash function	https://en.wikipedia.org/wiki/Cryptographic_hash_function	
10. SHA1	https://en.wikipedia.org/wiki/SHA-1	
11. မသတ်မှတ်တော့ပါ	https://web.archive.org/web/20260207211148/https://shattered.io/	
12. hash function	https://en.wikipedia.org/wiki/Hash_function	
13. Commitment schemes	https://en.wikipedia.org/wiki/Commitment_scheme	
14. key derivation functions	https://en.wikipedia.org/wiki/Key_derivation_function	
15. salt	https://en.wikipedia.org/wiki/Salt_(cryptography	

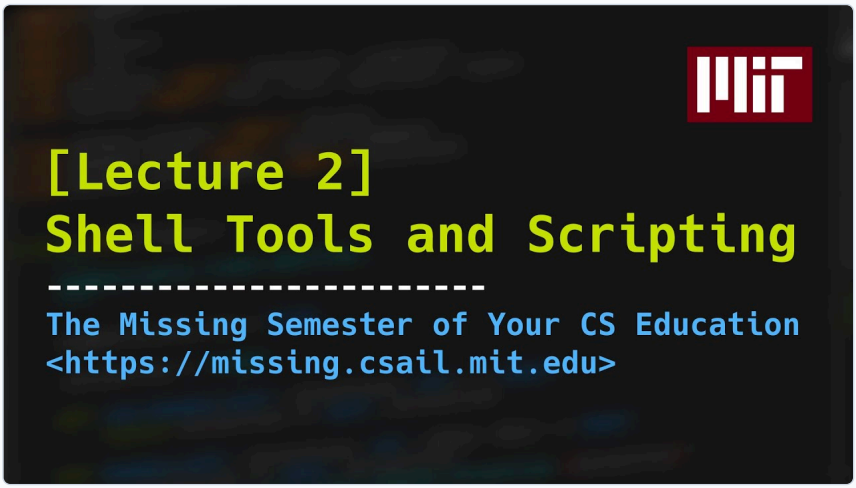
Resource / Description	URL	Micro QR
16. AES	https://en.wikipedia.org/wiki/Advanced_Encryption_Standard	
17. PGP email encryption	https://en.wikipedia.org/wiki/Pretty_Good_Privacy	
18. Signal	https://signal.org/	
19. Keybase	https://keybase.io/	
20. web of trust	https://en.wikipedia.org/wiki/Web_of_trust	
21. social proof	https://keybase.io/blog/chat-apps-softer-than-tofu	
22. KeePassXC	https://keepassxc.org/	
23. pass	https://git.zx2c4.com/password-store/about/	
24. 1Password	https://1password.com	
25. Two-factor authentication	https://en.wikipedia.org/wiki/Multi-factor_authentication	
26. cryptsetup + LUKS	https://wiki.archlinux.org/index.php/Dm-crypt/Encrypting_a_non-root_file_system	
27. BitLocker	https://fossbytes.com/enable-full-disk-encryption-windows-10/	
28. FileVault	https://support.apple.com/en-us/HT204837	
29. Cryptographic Right Answers	https://latacora.micro.blog/2018/04/03/cryptographic-right-answers.html	
30. Mirror	https://www.debian.org/CD/http-ftp/	

Shell Tools နှင့် Scripting

အနှစ်ချုပ် - Shell scripts များ ရေးသားနည်းနှင့် စွမ်းအားထက်မြက်သော Command-line tool များကို အသုံးပြုနည်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Shell Tools နှင့် Scripting



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=kgII-YWo3Zw

ဤသင်ခန်းစာတွင် bash ကို Scripting ဘာသာစကားအဖြစ် အသုံးပြုခြင်း အခြေခံများနှင့်အတူ Command-line တွင် မကြာခဏ ပြုလုပ်ရလေ့ရှိသော လုပ်ငန်းဆောင်တာများအတွက် အရေးပါသည့် Shell tool အများအပြားကို မိတ်ဆက်ပေးသွားမည် ဖြစ်ပါသည်။

Shell Scripting

ယခင် သင်ခန်းစာတွင် Shell တွင် Command များကို မည်သို့ Execute လုပ်ရမည်နှင့် ၎င်းတို့ကို Pipe ဖြင့် မည်သို့ ချိတ်ဆက်ရမည်ကို လေ့လာခဲ့ပြီး ဖြစ်သည်။ သို့သော် အခြေအနေ အများစုတွင် Command အများအပြားကို အစဉ်လိုက် လုပ်ဆောင်ရန်နှင့် Conditionals သို့မဟုတ် Loops ကဲ့သို့သော Control Flow များ အသုံးပြုရန် လိုအပ်လေ့ရှိသည်။

Shell scripts များသည် ပိုမို ရှုပ်ထွေးသော နောက်တစ်ဆင့် ဖြစ်ကြသည်။ Shell အများစုတွင် Variables များ၊ Control Flow များနှင့် သီးသန့် Syntax များ ပါဝင်သော ကိုယ်ပိုင် Scripting ဘာသာစကား ရှိကြသည်။ Shell Scripting ၏ အခြား Scripting ဘာသာစကားများနှင့် ကွဲပြားသော အချက်မှာ Shell နှင့် သက်ဆိုင်သော လုပ်ငန်းဆောင်တာများကို အထူး ပြုလုပ်နိုင်ရန် Design ထုတ်ထားခြင်း ဖြစ်သည်။ ထို့ကြောင့် Command Pipeline များ ဖန်တီးခြင်း၊ ရလဒ်များကို File များအဖြစ် သိမ်းဆည်းခြင်းနှင့် Standard Input မှ ဖတ်ရှုခြင်း တို့မှာ Shell Scripting တွင် မူလ ပါဝင်ပြီးသား ဖြစ်သဖြင့် ယေဘုယျ Scripting ဘာသာစကားများထက် အသုံးပြုရ ပိုမို လွယ်ကူစေပါသည်။ ဤသင်ခန်းစာတွင် အသုံးအများဆုံး ဖြစ်သည့် Bash Scripting ကို အဓိကထား လေ့လာသွားပါမည်။

Bash တွင် Variable များ သတ်မှတ်ရန် `foo=bar` syntax ကို အသုံးပြုပြီး Variable ၏ တန်ဖိုးကို ရယူရန် `$foo` ကို အသုံးပြုသည်။ `foo = bar` ဟု ရေးသားပါက အလုပ်လုပ်မည် မဟုတ်ပါ (ယင်းကို `foo` ပရိုဂရမ်အား `=` နှင့် `bar` အား Argument အဖြစ် ပေးပို့လိုက်သည်ဟု အဓိက အဓိပ္ပာယ်ဖော်သောကြောင့် ဖြစ်သည်)။ ယေဘုယျအားဖြင့် Shell Script များတွင် Space (ဟာကွက်) သည် Argument များကို ခွဲခြား ပေးသည်ကို သတိပြုပါ။

Bash တွင် String များကို `'` နှင့် `"` တို့ဖြင့် သတ်မှတ်နိုင်သော်လည်း ၎င်းတို့သည် တူညီမှု မရှိပါ။ `'` ဖြင့် သတ်မှတ်ထားသော String များသည် Literal String (မူရင်း စာသားအတိုင်း) ဖြစ်ပြီး Variable တန်ဖိုးများကို အစားထိုးပေးမည် မဟုတ်ဘဲ၊ `"` ဖြင့် သတ်မှတ်ထားသော String များတွင်မူ Variable တန်ဖိုးများကို အစားထိုးပေးမည် ဖြစ်သည်။

```
foo=bar
echo "$foo"
# prints bar
echo '$foo'
# prints $foo
```

အခြား ပရိုဂရမ်းမင်း ဘာသာစကားများကဲ့သို့ပင် Bash သည် `if` ၊ `case` ၊ `while` နှင့် `for` အပါအဝင် Control flow စနစ်များကို ပံ့ပိုးပေးသည်။ ထို့ပြင် `bash` တွင် Argument များကို လက်ခံပြီး အလုပ်လုပ်နိုင်သော Function များလည်း ပါရှိသည်။ အောက်ပါ ဥပမာမှာ Directory တစ်ခု ဖန်တီးပြီး ထို Directory အတွင်းသို့ `cd` ဝင်ရောက်ပေးသော Function ဖြစ်သည်-

```
mcd () {
    mkdir -p "$1"
    cd "$1"
}
```

ဒီနေရာမှာ `$1` သည် Script/Function ၏ ပထမဆုံး Argument ဖြစ်သည်။ အခြား Scripting ဘာသာစကားများနှင့် မတူဘဲ Bash တွင် Argument များ၊ Error Code များနှင့် အခြား သက်ဆိုင်ရာ Variable များကို ညွှန်ဆိုရန် အထူး Variable များကို အသုံးပြုကြသည်။ အောက်တွင် အချို့ကို စုစည်းဖော်ပြထားပါသည်- - `$0` - Script ၏ အမည် - `$1` မှ `$9` - Script သို့ ပေးပို့လိုက်သော Argument များ။ `$1` မှာ ပထမဆုံး Argument ဖြစ်သည်။ - `$@` - Argument အားလုံး - `$#` - Argument အရေအတွက် - `$?` - ယခင် Command ၏ Return Code (အလုပ်လုပ်ပုံ အခြေအနေ) - `$$` - လက်ရှိ Script ၏ Process Identification Number (PID) - `!!` - Argument အပါအဝင် ယခင် Command အပြည့်အစုံ။ မကြာခဏ အသုံးပြုသော စံနမူနာမှာ Permission မရှိ၍ Command ပျက်စီးသွားသည့်အခါ `sudo !!` ဖြင့် Command ကို sudo အဖြစ် ချက်ချင်း ပြန်လည် Run ခြင်း ဖြစ်သည်။ - `$_` - ယခင် Command ၏ နောက်ဆုံး Argument။ Interactive shell တွင် `Esc` နှိပ်ပြီး `.` သို့မဟုတ် `Alt+.` နှိပ်၍လည်း ဤတန်ဖိုးကို ရယူနိုင်သည်။

Command များသည် ပုံမှန်အားဖြင့် ရလဒ်များကို `STDOUT` မှလည်းကောင်း၊ Error များကို `STDERR` မှလည်းကောင်း ထုတ်ပေးကြပြီး၊ Error အခြေအနေများကို Script တွင် ပိုမို အဆင်ပြေစွာ စစ်ဆေးနိုင်ရန် Return Code များကို ပေးပို့ကြသည်။ Return Code သို့မဟုတ် Exit Status သည် Script/Command များ၏ လုပ်ဆောင်ချက် မည်သို့ ပြီးစီးခဲ့ကြောင်း ဖော်ပြသော နည်းလမ်းဖြစ်သည်။ တန်ဖိုး 0 ဖြစ်ပါက အရာအားလုံး အဆင်ပြေကြောင်း အဓိပ္ပာယ်ရပြီး 0 မဟုတ်ပါက Error ဖြစ်ပွားခဲ့ကြောင်း အဓိပ္ပာယ်ရသည်။

Exit Code များကို အသုံးပြု၍ Command များကို `&&` (AND operator) နှင့် `||` (OR operator) တို့ဖြင့် အခြေအနေပေါ်မူတည်၍ Run နိုင်ပါသည်။ Command များကို စာကြောင်း တစ်ကြောင်းတည်းတွင် မာလတီကော်လံ `;` ဖြင့်လည်း ခွဲခြားနိုင်ပါသည်။ `true` ပရိုဂရမ်သည် အမြဲတမ်း 0 return code ပေးပြီး `false` command သည် အမြဲတမ်း 1 return code ပေးသည်။

```
false || echo "Oops, fail"
# Oops, fail

true || echo "Will not be printed"
#

true && echo "Things went well"
# Things went well

false && echo "Will not be printed"
#

true ; echo "This will always run"
# This will always run

false ; echo "This will always run"
# This will always run
```

အခြား အသုံးများသော စံနမူနာတစ်ခုမှာ Command ၏ Output ကို Variable အဖြစ် ရယူလိုခြင်း ဖြစ်သည်။ ယင်းကို *Command substitution* ဖြင့် ပြုလုပ်နိုင်ပါသည်။ `$( CMD )` ဟု ရေးသားပါက `CMD` ကို Execute လုပ်ပြီး Output ကို ထိုနေရာတွင် အစားထိုးပေးမည် ဖြစ်သည်။ ဥပမာ `for file in $(ls)` ဟု ရေးပါက Shell က ပထမဦးစွာ `ls` ကို ခေါ်ယူပြီး ရရှိလာသော တန်ဖိုးများကို တိုင်ပတ် (iterate) လုပ်ဆောင်မည် ဖြစ်သည်။ အလားတူ လုပ်ဆောင်ချက်တစ်ခုမှာ *Process substitution* ဖြစ်ပြီး၊ `<( CMD )` သည် `CMD` ကို Execute လုပ်၍ Output ကို ယာယီ ဖိုင်အဖြစ် သိမ်းဆည်းကာ `<()` နေရာတွင် ထို ဖိုင်အမည်ကို အစားထိုးပေးသည်။ ဥပမာ `diff <(ls foo) <(ls bar)` သည် `foo` နှင့် `bar` directory ထဲရှိ file အပြောင်းအလဲများကို နှိုင်းယှဉ်ပြသပေးမည် ဖြစ်သည်။

အောက်ပါ ဥပမာတွင် Argument အဖြစ် ပေးပို့ထားသော ဖိုင်များကို စစ်ဆေး၍ `foobar` စာသား ပါရှိခြင်း မရှိပါက ဖိုင်၏ အဆုံးတွင် Comment အဖြစ် ထပ်ပေါင်းထည့်ပေးပုံကို ဖော်ပြထားသည်-

```
#!/bin/bash

echo "Starting program at $(date)" # Date will be substituted

echo "Running program $0 with $# arguments with pid $$"

for file in "$@"; do
    grep foobar "$file" > /dev/null 2> /dev/null
    # When pattern is not found, grep has exit status 1
    # We redirect STDOUT and STDERR to a null register since we do not care abo
ut them
    if [[ $? -ne 0 ]]; then
        echo "File $file does not have any foobar, adding one"
        echo "# foobar" >> "$file"
    fi
done
```

Bash တွင် နှိုင်းယှဉ်ချက်များ ပြုလုပ်ရာတွင် `[ ]` အစား Bracket နှစ်ထပ် `[[ ]]` ကို အသုံးပြုရန် အကြံပြုပါသည်။

Script များကို Run သည့်အခါ အလားတူ Argument များကို လွယ်ကူစွာ ပေးပို့နိုင်ရန် ဖိုင်အမည် တိုးချဲ့ခြင်း (Filename expansion) သို့မဟုတ် Shell *globbing* ကို အသုံးပြုနိုင်သည်- - Wildcards - `?` (စာလုံး တစ်လုံး) နှင့် `*` (စာလုံး မည်မျှမဆို) ကို အသုံးပြု၍ ရှာဖွေနိုင်ပါသည်။ ဥပမာ `rm foo?` သည် `foo1` နှင့် `foo2` ကို ဖျက်ဆီးမည် ဖြစ်ပြီး `rm foo*` သည် `bar` မှလွဲ၍ ကျန်ရှိသော `foo` စတင်သည့် ဖိုင်အားလုံးကို ဖျက်ဆီးမည် ဖြစ်သည်။ - Curly braces `{}` - တွန့်ကွင်း `{}` များကို အသုံးပြု၍ အသုံးများသော စာသားများကို အလိုအလျောက် တိုးချဲ့ခိုင်းနိုင်ပါသည်။ ဥပမာ `convert image.{png,jpg}` သည် `convert image.png image.jpg` ဟု တိုးချဲ့သွားမည် ဖြစ်သည်။

```
convert image.{png,jpg}
# Will expand to
convert image.png image.jpg

cp /path/to/project/{foo,bar,baz}.sh /newpath
# Will expand to
cp /path/to/project/foo.sh /path/to/project/bar.sh /path/to/project/baz.sh /new
path

# Globbing techniques can also be combined
mv *{.py,.sh} folder
# Will move all *.py and *.sh files

mkdir foo bar
# This creates files foo/a, foo/b, ... foo/h, bar/a, bar/b, ... bar/h
touch {foo,bar}/{a..h}
touch foo/x bar/y
# Show differences between files in foo and bar
diff <(ls foo) <(ls bar)
```

`bash` script များ ရေးသားရာတွင် အမှားများကို ကူညီ ရှာဖွေပေးရန် **shellcheck** (https://github.com/koalaman/shellcheck) ကဲ့သို့သော tool များကို အသုံးပြုနိုင်ပါသည်။

Script များကို Bash တစ်ခုတည်း မဟုတ်ဘဲ Python ကဲ့သို့သော အခြား ဘာသာစကားများဖြင့်လည်း ရေးသားနိုင်ပါသည်။ ဖိုင်၏ ထိပ်ဆုံးတွင် **Shebang** (https://en.wikipedia.org/wiki/Shebang_(Unix)) စာကြောင်း (ဥပမာ `#!/usr/bin/env python`) ထည့်သွင်းပေးထားပါက Kernel က သက်ဆိုင်ရာ Interpreter ဖြင့် Run ပေးမည် ဖြစ်သည်။

Shell Tools

Command များ အသုံးပြုပုံကို ရှာဖွေခြင်း (Finding how to use commands)

Command များနှင့် ပတ်သက်၍ အသေးစိတ် ညွှန်းဆိုချက် စာအုပ်များကို ကြည့်ရှုရန် `man` (https://www.man7.org/linux/man-pages/man1/man.1.html) (manual) ကို အသုံးပြုနိုင်ပါသည်။ ဥပမာ `man rm` ဖြင့် `rm` command ၏ အသုံးပြုပုံနှင့် Flag များကို ကြည့်ရှုနိုင်ပါသည်။

Manpage များသည် တစ်ခါတစ်ရံ လွန်စွာ အသေးစိတ်ကျသဖြင့် ရိုးရှင်းသော အသုံးပြုပုံ စံနမူနာများကို အမြန် ကြည့်လိုပါက **TLDR pages** (https://tldr.sh/) ကို အသုံးပြုနိုင်ပါသည်။

ဖိုင်များ ရှာဖွေခြင်း (Finding files)

UNIX စနစ်များတွင် ဖိုင်များနှင့် Directory များကို လိုက်လံ ရှာဖွေရန် **`find`** (https://www.man7.org/linux/man-pages/man1/find.1.html) tool ပါဝင်ပါသည်။

```
# Find all directories named src
find . -name src -type d
# Find all python files that have a folder named test in their path
find . -path '*/test/*.py' -type f
# Find all files modified in the last day
find . -mtime -1
# Find all zip files with size in range 500k to 10M
find . -size +500k -size -10M -name '*.tar.gz'
```

`find` သည် တွေ့ရှိသော ဖိုင်များပေါ်တွင် `-exec` Flag ဖြင့် Command များ တိုက်ရိုက် Run ပေးနိုင်ပါသည်။

`find` ၏ အစားထို ပိုမို မြန်ဆန် အသုံးပြုရ လွယ်ကူသော Tool အဖြစ် **`fd`** (https://github.com/sharkdp/fd) ကို အသုံးပြုနိုင်ပါသည်။ အလားတူ စနစ်အတွင်း အညွှန်း (index) စနစ်ဖြင့် လျှင်မြန်စွာ ရှာဖွေရန် **`locate`** (https://www.man7.org/linux/man-pages/man1/locate.1.html) ကို အသုံးပြုနိုင်ပါသည်။

ကုဒ်များ ရှာဖွေခြင်း (Finding code)

ဖိုင် စာသား အကြောင်းအရာများအတွင်း စာသား ပုံစံ (pattern) များကို ရှာဖွေရန် `grep` (https://www.man7.org/linux/man-pages/man1/grep.1.html) ကို အသုံးပြုသည်။ ပိုမို မြန်ဆန်သော ခေတ်မီ အစားထိုး Tool များအဖြစ် ripgrep (`rg`) (https://github.com/BurntSushi/ripgrep) ကို အသုံးပြုနိုင်သည်-

```
# Find all python files where I used the requests library
rg -t py 'import requests'
# Find all files (including hidden files) without a shebang line
rg -u --files-without-match "^#\!"
# Find all matches of foo and print the following 5 lines
rg foo -A 5
# Print statistics of matches (# of matched lines and files )
rg --stats PATTERN
```

Shell Command များကို ပြန်လည်ရှာဖွေခြင်း (Finding shell commands)

ယခင် ရိုက်ထည့်ခဲ့ဖူးသော Command များကို ရှာဖွေရန် `history` command ကို သို့မဟုတ် `Ctrl+R` ကို အသုံးပြုနိုင်ပါသည်။ ထို့ပြင် fzf (https://github.com/junegunn/fzf) fuzzy finder ကို အသုံးပြု၍ လည်းကောင်း၊ fish (https://fishshell.com/) သို့မဟုတ် zsh-autosuggestions (https://github.com/zsh-users/zsh-autosuggestions) ဖြင့် History အခြေပြု အလိုအလျောက် အကြံပြုချက်များ ရယူ၍လည်းကောင်း အသုံးပြုနိုင်ပါသည်။

Directory များအတွင်း သွားလာခြင်း (Directory Navigation)

မကြာခဏ သွားရောက်လေ့ရှိသော Directory များသို့ လျင်မြန်စွာ `cd` ဝင်ရောက်ရန် `fasd` (https://github.com/clvv/fasd) (command `z`) သို့မဟုတ် `autojump` (https://github.com/wting/autojump) (command `j`) တို့ကို အသုံးပြုနိုင်ပါသည်။ Directory အဆောက်အအုံကို သtree ပုံစံ ကြည့်ရှုရန် `tree` (https://linux.die.net/man/1/tree) သို့မဟုတ် `broot` (https://github.com/Canop/broot) တို့ကို အသုံးပြုနိုင်ပါသည်။

လေ့ကျင့်ခန်းများ (Exercises)

၁။ `man ls` (https://www.man7.org/linux/man-pages/man1/ls.1.html) ကို ဖတ်ရှုပြီး ဖိုင်ဝှက်များ ပါဝင်သော၊ ဖိုင်ပမာဏကို ဖတ်ရှုရ လွယ်ကူသော format (ဥပမာ 454M) ဖြင့် ဖော်ပြထားသော၊ မကြာသေးမီက ပြင်ဆင်ခဲ့သော စီစဉ်မှုအတိုင်း စီထားသော အရောင်ပါဝင်သည့် `ls` command တစ်ခု ရေးသားပါ။

၂။ `marco` နှင့် `polo` ဟုခေါ်သော Bash function နှစ်ခုကို ရေးသားပါ။ `marco` ကို Execute လုပ်သည့် အခါ လက်ရှိ ရောက်ရှိနေသော Directory ကို သိမ်းဆည်းထားရမည်ဖြစ်ပြီး၊ မည်သည့် Directory တွင် ရောက်ရှိနေသည်ဖြစ်စေ `polo` ကို Execute လုပ်လိုက်သည်နှင့် `marco` Run ခဲ့သော Directory သို့ ပြန်လည် `cd` ဝင်ရောက်သွားရမည် ဖြစ်သည်။

၃။ ပျက်စီးခဲသော Command တစ်ခု ရှိသည် ဆိုပါစို့။ ထို Command ပျက်စီးသွားသည့်အထိ ထပ်ခါတလဲလဲ Run ပေးပြီး Error နှင့် Output များကို ဖိုင်ထဲသို့ သိမ်းဆည်းကာ မည်မျှ ကြိမ်ဖန် Run ခဲ့ရကြောင်း ထုတ်ပြန်ပေးသော Bash script တစ်ခု ရေးသားပါ။

```bash
#!/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"
```

၄။ `find` ၏ `-exec` ကို အသုံးပြု၍ ရှာဖွေတွေ့ရှိသမျှ HTML ဖိုင်များအားလုံးကို Zip ဖိုင်အဖြစ် ပြောင်းလဲပေးသော Command တစ်ခုကို `xargs` (https://www.man7.org/linux/man-pages/man1/xargs.1.html) အသုံးပြု၍ ရေးသားပါ။

၅။ (အဆင့်မြင့်) Directory တစ်ခုအတွင်း မကြာသေးမီက ပြင်ဆင်ထားသော ဖိုင်ကို ရှာဖွေပေးသည့် သို့မဟုတ် ဖိုင်အားလုံးကို ပြင်ဆင်ခဲ့သော အချိန်အလိုက် စီစဉ်ပေးသည့် Command သို့မဟုတ် Script တစ်ခု ရေးသားပါ။

🔗 External Links & References (ပြင်ပ လင့်ခ်များနှင့် ကိုးကားချက်များ)

Resource / Description	URL	Micro QR
1. shellcheck	https://github.com/koalaman/shellcheck	
2. Shebang	https://en.wikipedia.org/wiki/Shebang_(Unix	
3. `man`	https://www.man7.org/linux/man-pages/man1/man.1.html	
4. TLDR pages	https://tldr.sh/	
5. `find`	https://www.man7.org/linux/man-pages/man1/find.1.html	
6. `fd`	https://github.com/sharkdp/fd	
7. `locate`	https://www.man7.org/linux/man-pages/man1/locate.1.html	
8. `grep`	https://www.man7.org/linux/man-pages/man1/grep.1.html	
9. ripgrep (`rg`)	https://github.com/BurntSushi/ripgrep	
10. fzf	https://github.com/junegunn/fzf	
11. fish	https://fishshell.com/	
12. zsh-autosuggestions	https://github.com/zsh-users/zsh-autosuggestions	
13. `fasd`	https://github.com/clvv/fasd	
14. `autojump`	https://github.com/wting/autojump	
15. `tree`	https://linux.die.net/man/1/tree	

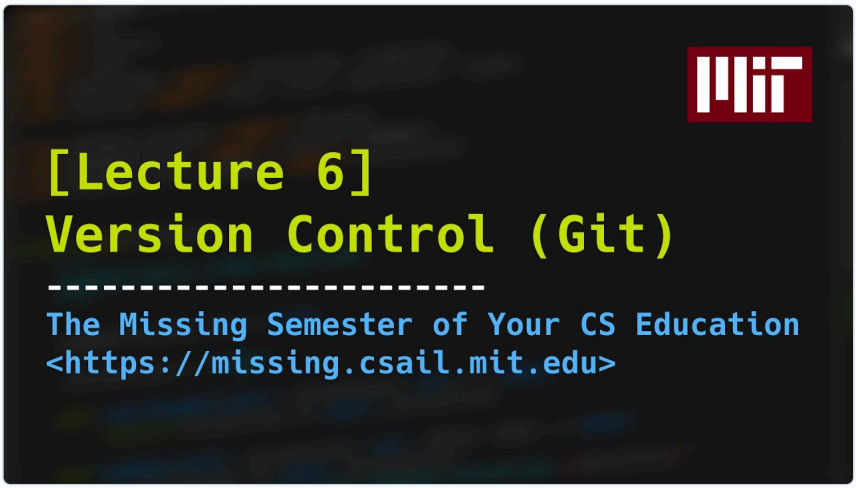
Resource / Description	URL	Micro QR
16. `broot`	https://github.com/Canop/broot	
17. `man ls`	https://www.man7.org/linux/man-pages/man1/ls.1.html	
18. `xargs`	https://www.man7.org/linux/man-pages/man1/xargs.1.html	

Version Control (Git)

အနှစ်ချုပ် - Git ၏ data model နှင့် Version control၊ ပူးပေါင်းဆောင်ရွက်မှုများအတွက် Git ကို မည်သို့ အသုံးပြုရမည်နည်း လေ့လာပါ။

▶ LECTURE VIDEO REFERENCE

Version Control (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=2sjqTHE0zok

Version control systems (VCSs) ဆိုသည်မှာ Source Code များ (သို့မဟုတ် အခြား ဖိုင်နှင့် Folder စုစည်းမှုများ) ၏ အပြောင်းအလဲများကို ခြေရာခံရန် အသုံးပြုသည့် Tool များ ဖြစ်ကြသည်။ အမည်တွင် ဖော်ပြထားသည့်အတိုင်း ဤ Tool များသည် အပြောင်းအလဲများ၏ သမိုင်းကြောင်းကို ထိန်းသိမ်းပေးသည့်အပြင် အခြားသူများနှင့် ပူးပေါင်းဆောင်ရွက်မှုကိုလည်း လွယ်ကူစေပါသည်။ VCS များသည် Folder တစ်ခုနှင့် ယင်း၏ ပါဝင်မှုများကို ပုံရိပ်လွှာ (snapshot) အစဉ်လိုက်များဖြင့် ခြေရာခံပေးပြီး၊ Snapshot တစ်ခုစီသည် Top-level Directory အတွင်းရှိ ဖိုင်/Folder များ၏ အခြေအနေတစ်ခုလုံးကို ဆုံကြည့်နိုင်အောင် စုစည်းထားပေးပါသည်။ ထို့အပြင် VCS များသည် Snapshot တစ်ခုစီကို မည်သူဖန်တီးခဲ့သည်၊ အမှတ်အသား စာတိုများ (messages) အစရှိသည့် Metadata များကိုပါ ထိန်းသိမ်းပေးပါသည်။

Version control ကို အသုံးပြုခြင်းသည် အဘယ်ကြောင့် အသုံးဝင်သနည်း။ မိမိတစ်ဦးတည်း အလုပ်လုပ်နေချိန်တွင်ပင် ပရောဂျက်၏ Snapshot အဟောင်းများကို ပြန်လည်ကြည့်ရှုနိုင်ခြင်း၊ အချို့သော အပြောင်းအလဲများကို အဘယ်ကြောင့် ပြုလုပ်ခဲ့ကြောင်း မှတ်တမ်းတင်ထားနိုင်ခြင်း၊ ပြီးတော့ ဖွံ့ဖြိုးတိုးတက်ရေး Branch များ (parallel branches) ဖြင့် အလုပ်လုပ်နိုင်ခြင်း အစရှိသည်တို့ကို ပြုလုပ်နိုင်ပါသည်။ အခြားသူများနှင့် ပူးပေါင်းလုပ်ဆောင်သည့်အခါတွင်လည်း အခြားသူများ မည်သည့်အရာများ ပြောင်းလဲထားသည်ကို ကြည့်ရှုနိုင်သည့်အပြင် ပြီးတော့ လုပ်ဆောင်ရာတွင် ဖြစ်ပေါ်လာသည့် ပဋိပက္ခများ (conflicts) ကို ဖြေရှင်းရာ၌ အလွန်တန်ဖိုးရှိသော Tool ဖြစ်ပါသည်။

ခေတ်မီ VCS များသည် အောက်ပါ မေးခွန်းများကိုလည်း လွယ်ကူစွာ (နှင့် အလိုအလျောက်) ဖြေကြားပေးနိုင်ပါသည် -

- ဤ Module ကို မည်သူ ရေးသားခဲ့သနည်း။
- ဤ ဖိုင်၏ ဤ သီးခြား စာကြောင်းကို မည်သည့်အချိန်တွင်၊ မည်သူက ပြင်ဆင်ခဲ့သနည်း။ အဘယ်ကြောင့် ပြင်ဆင်ခဲ့သနည်း။
- လွန်ခဲ့သော မူကွဲ ၁၀၀၀ အတွင်း၊ မည်သည့်အချိန်/မည်သည့်အကြောင်းကြောင့် သီးခြား Unit Test တစ်ခု အလုပ်မလုပ်တော့ဘဲ ဖြစ်သွားသနည်း။

အခြားသော VCS များ ရှိသော်လည်း **Git** သည် Version control အတွက် အဓိက စံနှုန်း (de facto standard) ဖြစ်ပါသည်။ ဤ **XKCD comic** (https://xkcd.com/1597/) လေးသည် Git ၏ ကျော်ကြားမှုကို သရော်ထားပါသည် -

!**xkcd 1597** (https://imgs.xkcd.com/comics/git.png)

Git ၏ Interface သည် leaky abstraction ဖြစ်သောကြောင့် Git ကို Top-down နည်းလမ်းဖြင့် (ယင်း၏ Interface / Command-line Interface မှ စတင်၍) လေ့လာခြင်းသည် ရှုပ်ထွေးမှုများစွာ ဖြစ်ပေါ်စေနိုင်ပါသည်။ Command အနည်းငယ်ကို အလွတ်ကျက်မှတ်ထားပြီး မန္တန်အဖြစ် မှတ်ယူကာ၊ ပြဿနာ တစ်ခုခု တက်လာပါက အထက်ပါ Comic ပါ အယူအဆအတိုင်း ပြုလုပ်လေ့ ရှိကြပါသည်။

Git တွင် ရှုပ်ထွေးသော Interface ရှိသည်ကို လက်ခံရမည် ဖြစ်သော်လည်း ယင်း၏ အောက်ခံ ဒီဇိုင်းနှင့် အယူအဆများသည် အလွန် သပ်ရပ်လှပပါသည်။ ရှုပ်ထွေးသော Interface ကို အလွတ်ကျက်မှတ်ရမည် ဖြစ်သော်လည်း လှပသော ဒီဇိုင်းကိုမူ *နားလည်သဘောပေါက်* နိုင်ပါသည်။ ထို့ကြောင့် ကျွန်ုပ်တို့သည် Git ကို ယင်း၏ Data model မှ စတင်၍ အောက်ခြေမှ အထက်သို့ (Bottom-up) ရှင်းလင်းသင်ကြားပေးမည် ဖြစ်ပြီး၊ နောက်ပိုင်းမှ Command-line Interface အကြောင်းကို လွှမ်းခြုံ သင်ကြားပါမည်။ Data model ကို နားလည်သွားပါက Command များသည် အောက်ခံ Data model ကို မည်သို့ စီမံခန့်ခွဲသည်ဆိုသည်ကို ပိုမို နားလည်သဘောပေါက်လာပါမည်။

Git’s data model

Version control အတွက် အသုံးပြုနိုင်သော နည်းလမ်းများစွာ ရှိပါသည်။ Git တွင် Version control ၏ အင်္ဂါရုပ်ကောင်းများဖြစ်သော သမိုင်းကြောင်း ထိန်းသိမ်းခြင်း၊ Branch များကို ထောက်ပံ့ခြင်း နှင့် ပူးပေါင်းဆောင်ရွက်မှုကို လွယ်ကူစေခြင်း အစရှိသည်တို့ကို အပြည့်အဝ ပေးစွမ်းနိုင်သည့် သေချာစွာ စနစ်တကျ ရေးဆွဲထားသော Model တစ်ခု ရှိပါသည်။

Snapshots

Git သည် Top-level Directory တွင်းရှိ ဖိုင်များနှင့် Folder များ စုစည်းမှု၏ သမိုင်းကြောင်းကို Snapshot များအစဉ်လိုက်အဖြစ် ပုံဖော်ထားပါသည်။ Git ဝေါဟာရတွင် ဖိုင်ကို “blob” ဟု ခေါ်ဆိုပြီး ယင်းသည် Byte အစုအဝေးမျှသာ ဖြစ်သည်။ Directory ကို “tree” ဟု ခေါ်ဆိုပြီး ယင်းသည် အမည်များကို blobs သို့မဟုတ် trees သို့ ချိတ်ဆက်ပေးပါသည် (ထို့ကြောင့် Directory များတွင် အခြား Directory များ ပါဝင်နိုင်သည်)။ Snapshot ဆိုသည်မှာ ခြေရာခံနေသော Top-level Tree ဖြစ်ပါသည်။ ဥပမာအားဖြင့် အောက်ပါအတိုင်း Tree တစ်ခု ရှိနိုင်ပါသည် -

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

Top-level Tree တွင် အစိတ်အပိုင်း နှစ်ခု ပါဝင်ပါသည်၊ “foo” ဟုခေါ်သော Tree တစ်ခု (၎င်းကိုယ်တိုင်တွင် “bar.txt” ဟုခေါ်သော blob တစ်ခု ပါဝင်သည်) နှင့် “baz.txt” ဟုခေါ်သော blob တစ်ခုတို့ ဖြစ်ကြသည်။

Modeling history: relating snapshots

Version control စနစ်တစ်ခုသည် Snapshot များကို မည်သို့ ဆက်စပ်ပေးသင့်သနည်း။ ရိုးရှင်းသော Model တစ်ခုမှာ မျဉ်းဖြောင့် သမိုင်းကြောင်း (linear history) ဖြစ်ပါမည်။ သမိုင်းကြောင်းသည် အချိန်အစဉ်လိုက် Snapshot များ၏ စာရင်း ဖြစ်ပါမည်။ သို့သော် အကြောင်းကြောင်း ကြောင့် Git သည် ဤကဲ့သို့ ရိုးရှင်းသော Model ကို မသုံးပါ။

Git တွင် သမိုင်းကြောင်းဆိုသည်မှာ Snapshot များ၏ Directed Acyclic Graph (DAG) ဖြစ်ပါသည်။ ဤသည်မှာ သခ်ၤာစကားလုံးဆန်း ဖြစ်ကောင်းဖြစ်နိုင်သော်လည်း စိတ်မပူပါနှင့်။ ဤသည်၏ အဓိပ္ပာယ်မှာ Git ရှိ Snapshot တစ်ခုစီသည် ယင်း၏ အလျင် ရှိခဲ့သော မိဘများ “parents” ၏ အစုအဝေးကို ညွှန်းဆိုနေခြင်း ဖြစ်ပါသည်။ မိဘ တစ်ခုတည်း (linear history တွင် ဖြစ်သကဲ့သို့) မဟုတ်ဘဲ မိဘ အစုအဝေး ဖြစ်နေရခြင်းမှာ Snapshot တစ်ခုသည် မိဘများစွာမှ ဆင်းသက်လာနိုင်သောကြောင့် ဖြစ်သည် (ဥပမာ ပြိုင်တူ ဖွံ့ဖြိုးတိုးတက်နေသော Branch နှစ်ခုကို ပေါင်းစပ် merge လုပ်လိုက်သည့်အခါမျိုးတွင် ဖြစ်သည်)။

Git သည် ဤ Snapshot များကို “commit” များဟု ခေါ်ဆိုပါသည်။ Commit သမိုင်းကြောင်းကို ပုံဖော်ကြည့်ပါက အောက်ပါအတိုင်း တွေ့ရပါမည် -

```
o <-- o <-- o <-- o
            ^
             \
              --- o <-- o
```

အထက်ပါ ASCII Art တွင် `o` များသည် သီးခြား Commit (snapshot) များကို ကိုယ်စားပြုပါသည်။ မြား များသည် Commit တစ်ခုစီ၏ Parent ကို ညွှန်ပြနေခြင်း ဖြစ်ပါသည် (“နောက်မှ လာသည်” မဟုတ်ဘဲ “ရှေ့မှ လာသည်” ဆက်ဆံရေး ဖြစ်သည်)။ တတိယ Commit ပြီးနောက် သမိုင်းကြောင်းသည် သီးခြား Branch နှစ်ခုအဖြစ် ခွဲထွက်သွားသည်။ ဤသည်မှာ ဥပမာအားဖြင့် သီးခြား Feature နှစ်ခုကို တစ်ပြိုင်နက်တည်း သီးခြားစီ ဖွံ့ဖြိုးတိုးတက်အောင် ပြုလုပ်နေခြင်းနှင့် တူညီပါသည်။ နောင်တွင် ဤ Branch နှစ်ခုကို Merge ပြုလုပ်၍ Feature နှစ်ခုစလုံး ပါဝင်သော Snapshot အသစ်တစ်ခုကို ဖန်တီးနိုင်ပြီး၊ အောက်ပါအတိုင်း သမိုင်းကြောင်း အသစ်တစ်ခု ထွက်ပေါ်လာမည် ဖြစ်ကာ Merge commit အသစ်ကို စာလုံးအထူဖြင့် ဖော်ပြထားပါသည် -

```
o <-- o <-- o <-- o <---- o
```bash
        ^           /
         \         v
          --- o <-- o
```
```

Git ရှိ Commit များသည် မပြောင်းလဲနိုင်သော (immutable) အရာများ ဖြစ်ကြသည်။ ဤသည်မှာ အမှားများကို ပြင်ဆင်၍ မရနိုင်ဟု ဆိုလိုခြင်း မဟုတ်ပါ၊ Commit သမိုင်းကြောင်းကို “ပြင်ဆင်ခြင်း” ဆိုသည်မှာ အမှန်တကယ်အားဖြင့် Commit အသစ်တစ်ခုလုံးကို ဖန်တီးလိုက်ခြင်း ဖြစ်ပြီး References များကို Commit အသစ်သို့ ညွှန်ပြအောင် အပ်ဒိတ်လုပ်လိုက်ခြင်း ဖြစ်သည်။

Data model, as pseudocode

Git ၏ Data model ကို Pseudocode ဖြင့် ရေးသားထားသည်ကို ကြည့်ပါက ပိုမို ရှင်းလင်းသွားပါမည် -

```
// file ဆိုသည်မှာ Byte အစုအဝေး ဖြစ်သည်
type blob = array<byte>

// directory တွင် အမည်တပ်ထားသော ဖိုင်များနှင့် directory များ ပါဝင်သည်
type tree = map<string, tree | blob>

// commit တွင် parents, metadata နှင့် top-level tree တို့ ပါဝင်သည်
type commit = struct {
    parents: array<commit>
    author: string
    message: string
    snapshot: tree
}
```

ဤသည်မှာ သမိုင်းကြောင်း၏ သပ်ရပ်ရှင်းလင်းသော Model ဖြစ်ပါသည်။

Objects and content-addressing

“object” ဆိုသည်မှာ blob, tree သို့မဟုတ် commit ဖြစ်ပါသည် -

```
type object = blob | tree | commit
```

Git data store တွင် object အားလုံးကို ယင်းတို့၏ **SHA-1 hash** (https://en.wikipedia.org/wiki/SHA-1) ဖြင့် Content-addressing ပြုလုပ်ထားပါသည် -

```
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 ပေါင်းစည်းထားပါသည် - ယင်းတို့ အားလုံးသည် object များ ဖြစ်ကြသည်။ ယင်းတို့သည် အခြား object များကို ညွှန်းဆိုသည့်အခါ၊ Disc ပေါ်တွင် Object တစ်ခုလုံးကို ထည့်သွင်း ထားခြင်း မဟုတ်ဘဲ Hash ဖြင့် ညွှန်းဆိုထားသော Reference ကိုသာ သိမ်းဆည်းထားပါသည်။

ဥပမာအားဖြင့် အထက်ပါ Directory စနစ်၏ Tree ကို (`git cat-file -p 698281bc680d1995c5f4caaf3359721a5a58d48d` ဖြင့် ကြည့်ရှုထားသော) အောက်ပါအတိုင်း တွေ့ရပါမည် -

```
100644 blob 4448adbf7ecd394f42ae135bbeed9676e894af85    baz.txt
040000 tree c68d233a33c5c06e0340e4c224f0afca87c8ce87    foo
```

Tree ကိုယ်တိုင်တွင် ပါဝင်သော အရာများဖြစ်သည့် `baz.txt` (blob) နှင့် `foo` (tree) သို့ ညွှန်ပြသော Pointers များ ပါဝင်ပါသည်။ `git cat-file -p 4448adbf7ecd394f42ae135bbeed9676e894af85` Command ဖြင့် `baz.txt` ၏ hash ဖြင့် ကြည့်ရှုလိုက်ပါက အောက်ပါအတိုင်း ရရှိမည် ဖြစ်သည် -

```
git is wonderful
```

References

ယခုအခါ Snapshot အားလုံးကို SHA-1 Hashes များဖြင့် ခွဲခြားသတ်မှတ်နိုင်ပြီ ဖြစ်သည်။ သို့သော် လူများသည် 16 စီးပါ 40-digit Hexadecimal စာလုံးများကို မှတ်မိရန် မလွယ်ကူသဖြင့် အဆင်မပြေပါ။

ဤပြဿနာအတွက် Git ၏ ဖြေရှင်းချက်မှာ SHA-1 Hashes များကို လူနားလည်လွယ်သော အမည်များ ပေးခြင်းဖြစ်ပြီး ယင်းတို့ကို “References” ဟု ခေါ်ဆိုပါသည်။ References များသည် Commit များကို ညွှန်ပြသော Pointers များ ဖြစ်ကြသည်။ မပြောင်းလဲနိုင်သော Objects များနှင့် မတူဘဲ References များသည် ပြောင်းလဲနိုင်ပါသည် (Commit အသစ်သို့ ညွှန်ပြအောင် အပ်ဒိတ်လုပ်နိုင်သည်)။ ဥပမာအားဖြင့် `master` reference သည် အဓိက Branch ၏ နောက်ဆုံး Commit ကို ညွှန်ပြလေ့ ရှိပါသည်။

```
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)
```

ဤနည်းဖြင့် Git သည် ရှည်လျားသော Hexadecimal String အစား “master” ကဲ့သို့သော လူနားလည်လွယ်သော အမည်များကို သမိုင်းကြောင်းရှိ သီးခြား Snapshot များကို ညွှန်းဆိုရန် အသုံးပြုနိုင်ပါသည်။

အသေးစိတ် အချက်တစ်ခုမှာ သမိုင်းကြောင်းတွင် “လက်ရှိ ရောက်ရှိနေသော နေရာ” ကို သိရှိလိုခြင်း ဖြစ်သည်၊ ထို့မှသာ Snapshot အသစ်တစ်ခု ရယူသည့်အခါ မည်သည့်အရာနှင့် ယှဉ်တွဲရမည်နည်း (Commit ၏ `parents` field ကို မည်သို့ သတ်မှတ်မည်နည်း) ကို သိရှိနိုင်မည် ဖြစ်သည်။ Git တွင် ထို “လက်ရှိ ရောက်ရှိနေသော နေရာ” သည် “HEAD” ဟုခေါ်သော အထူး Reference ဖြစ်ပါသည်။

Repositories

နောက်ဆုံးတွင် Git *repository* ၏ အဓိပ္ပာယ်ကို သတ်မှတ်နိုင်ပါပြီ - ယင်းသည် `objects` နှင့် `references` Data များ ဖြစ်ကြပါသည်။

Disc ပေါ်တွင် Git သိမ်းဆည်းထားသမျှသည် objects နှင့် references များသာ ဖြစ်ကြသည် - Git ၏ Data model တွင် ဒါအကုန်ပါပဲ။ `git` command အားလုံးသည် objects များကို ထည့်သွင်းခြင်းနှင့် references များကို အပ်ဒိတ်လုပ်ခြင်းဖြင့် Commit DAG ကို မွမ်းမံ ပြင်ဆင်နေခြင်း ဖြစ်ပါသည်။

Command တစ်ခုခုကို ရိုက်နှိပ်လိုက်တိုင်း၊ ထို Command သည် အောက်ခံ Graph Data Structure ကို မည်သို့ ပြောင်းလဲနေသည်ဆိုသည်ကို စဉ်းစားပါ။ အလားတူပင် Commit DAG သို့ သီးခြား ပြောင်းလဲမှုတစ်ခုခု ပြုလုပ်လိုပါက (ဥပမာ “မသိမ်းရသေးသော အပြောင်းအလဲများကို စွန့်ပစ်ပြီး ‘master’ ref ကို commit `5d83f9e` သို့ ညွှန်ပြပါ”) ထိုသို့ ပြုလုပ်ရန် Command တစ်ခုခု ရှိနေမည် ဖြစ်သည် (ဥပမာ `git checkout master; git reset --hard 5d83f9e`)။

Staging area

ဤသည်မှာ Data model နှင့် သီးခြား အယူအဆ ဖြစ်သော်လည်း Commit များ ဖန်တီးသည့် Interface ၏ အစိတ်အပိုင်းတစ်ခု ဖြစ်ပါသည်။

Snapshot ရယူခြင်းကို အကောင်အထည်ဖော်ရာတွင် Working Directory ၏ *လက်ရှိ အခြေအနေ* ကို အခြေခံ၍ Snapshot အသစ် ဖန်တီးပေးသည့် “create snapshot” command တစ်ခု ရှိမည်ဟု ထင်မြင်နိုင်ပါသည်။ အချို့သော Version control tool များသည် ဤသို့ အလုပ်လုပ်ကြသော်လည်း Git မဟုတ်ပါ။ ကျွန်ုပ်တို့သည် သပ်ရပ်သန့်ရှင်းသော Snapshot များကို အလိုရှိကြပြီး၊ လက်ရှိ အခြေအနေ တစ်ခုလုံးမှ Snapshot ရယူခြင်းသည် အမြဲတမ်း မသင့်တော်နိုင်ပါ။ ဥပမာအားဖြင့် သီးခြား Feature နှစ်ခုကို ရေးသားထားပြီး သီးခြား Commit နှစ်ခု ဖန်တီးလိုသည် ဆိုပါစို့၊ ပထမ Commit တွင် ပထမ Feature ပါဝင်ပြီး ဒုတိယ Commit တွင် ဒုတိယ Feature ပါဝင်စေလိုသည်။ သို့မဟုတ် Code တွင် ပုံနှိပ်ထုတ်ဝေထားသော Debugging print စာကြောင်းများ ပါဝင်နေပြီး Bugfix နှင့်အတူ ရောနှောနေသည် ဆိုပါစို့၊ Bugfix ကိုသာ Commit လုပ်ပြီး Print စာကြောင်းများကို စွန့်ပစ်လိုပါမည်။

Git သည် “Staging area” ဟုခေါ်သော နည်းလမ်းမှတဆင့် မည်သည့် အပြောင်းအလဲများကို နောက် Snapshot တွင် ထည့်သွင်းမည်ဆိုသည်ကို သီးခြား သတ်မှတ်ခွင့် ပြုထားပါသည်။

Git command-line interface

အချက်အလက်များ ထပ်မံ မဖြစ်စေရန်အတွက် အောက်ပါ Command များကို အသေးစိတ် ရှင်းပြမည် မဟုတ်ပါ။ အသေးစိတ်အတွက် အလွန် အကြံပြုထားသော **Pro Git** (https://git-scm.com/book/en/v2) ကို ဖတ်ရှုပါ သို့မဟုတ် သင်ခန်းစာ ဗီဒီယိုကို ကြည့်ရှုပါ။

Basics

- `git help <command>` : git command အတွက် အကူအညီ ရယူရန်
- `git init` : Git repo အသစ်တစ်ခု ဖန်တီးရန် (`.git` directory တွင် metadata များကို သိမ်းဆည်းသည်)
- `git status` : လက်ရှိ အခြေအနေများကို ကြည့်ရှုရန်
- `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 message များ</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 message များ ရေးသားရခြင်း အကြောင်းအရင်းများ</a> <span class="ext-url-print">(https://chris.beams.io/posts/git-commit/)</span>!
```

- `git log` : သမိုင်းကြောင်း မှတ်တမ်းများကို မျဉ်းဖြောင့် ပုံစံဖြင့် ပြသရန်
- `git log --all --graph --decorate` : သမိုင်းကြောင်းကို DAG Graph ပုံစံဖြင့် မြင်တွေ့ရရန်
- `git diff <filename>` : Staging area နှင့် ယှဉ်လျှင် ပြုလုပ်ထားသော အပြောင်းအလဲများကို ပြသရန်
- `git diff <revision> <filename>` : Snapshot များ ကြားရှိ ဖိုင်ပြောင်းလဲမှုများကို ပြသရန်
- `git checkout <revision>` : HEAD ကို အပ်ဒိတ်လုပ်ရန် (Branch ကို checkout လုပ်ပါက လက်ရှိ branch ကိုပါ ပြောင်းလဲပေးသည်)

Branching and merging

- `git branch` : Branch များကို ပြသရန်
- `git branch <name>` : Branch အသစ် ဖန်တီးရန်
- `git checkout -b <name>` : Branch အသစ် ဖန်တီးပြီး ထို branch သို့ ချက်ချင်း ပြောင်းလဲရန်

```
- `git branch <name>; git checkout <name>` နှင့် တူညီသည်
```

- `git merge <revision>` : လက်ရှိ branch အတွင်းသို့ ပေါင်းစပ်ရန်
- `git mergetool` : Merge conflicts များကို ဖြေရှင်းရန် Tool ဖြင့် အကူအညီ ရယူရန်
- `git rebase` : Patches များကို Base အသစ်တစ်ခု ပေါ်သို့ Rebase ပြုလုပ်ရန်

Remotes

- `git remote` : Remote စာရင်းများကို ပြရန်
- `git remote add <name> <url>` : Remote အသစ် ထည့်ရန်
- `git push <remote> <local branch>:<remote branch>` : Remote သို့ Objects များကို ပို့ပြီး Remote reference ကို အပ်ဒိတ်လုပ်ရန်
- `git branch --set-upstream-to=<remote>/<remote branch>` : Local နှင့် Remote branch ကြား ဆက်သွယ်မှု သတ်မှတ်ရန်
- `git fetch` : Remote မှ Objects/References များကို ယူဆောင်ရန်
- `git pull` : `git fetch; git merge` နှင့် တူညီသည်
- `git clone` : Remote မှ Repository ကို ဒေါင်းလုဒ် ရယူရန်

Undo

- `git commit --amend` : Commit ၏ စာတို သို့မဟုတ် အကြောင်းအရာကို ပြင်ဆင်ရန်
- `git reset HEAD <file>` : Staging area မှ ဖိုင်ကို ပြန်ထုတ်ရန်
- `git checkout -- <file>` : ပြုလုပ်ထားသော အပြောင်းအလဲများကို စွန့်ပစ်ရန်

Advanced Git

- `git config` : Git ကို စိတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်း မြင့်မားသည် (https://git-scm.com/docs/git-config)
- `git clone --depth=1` : သမိုင်းကြောင်း အပြည့်အစုံ မပါဘဲ အပေါ်ယံ Clone လုပ်ရန်
- `git add -p` : Interactive ပုံစံဖြင့် Staging ပြုလုပ်ရန်
- `git rebase -i` : Interactive ပုံစံဖြင့် Rebase ပြုလုပ်ရန်
- `git blame` : စာကြောင်း တစ်ကြောင်းစီကို မည်သူ နောက်ဆုံး ပြင်ဆင်ခဲ့သည်ကို ပြသရန်
- `git stash` : Working directory ရှိ ပြင်ဆင်ချက်များကို ခေတ္တ ဖယ်ရှားသိမ်းဆည်းထားရန်
- `git bisect` : သမိုင်းကြောင်းတွင် အမှားများကို Binary search ဖြင့် ရှာဖွေရန်
- `.gitignore` : ခြေရာမခံဘဲ ပစ်ပယ်ထားမည့် ဖိုင်များကို သတ်မှတ်ရန် (https://git-scm.com/docs/gitignore)

Miscellaneous

- **GUIs**: Git အတွက် **GUI clients** (https://git-scm.com/downloads/guis) များစွာ ရှိပါသည်။ ကျွန်ုပ်တို့ ကိုယ်တိုင်မှ ယင်းတို့ကို မသုံးဘဲ Command-line interface ကိုသာ အသုံးပြုပါသည်။
- **Shell integration**: သင်၏ Shell prompt တွင် 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 မူလ ပါဝင်လေ့ရှိသည်။
- **Editor integration**: အထက်ပါအတိုင်း အသုံးဝင်သော အင်္ဂါရုပ်များစွာဖြင့် ပေါင်းစပ်ထားနိုင်ပါသည်။ **fugitive.vim** (https://github.com/tpope/vim-fugitive) သည် Vim အတွက် စံနှုန်း ဖြစ်ပါသည်။
- **Workflows**: ကျွန်ုပ်တို့သည် Data model နှင့် အခြေခံ Command များကို သင်ကြားပေးခဲ့ပြီး ဖြစ်သည်; ပရောဂျက် ကြီးများတွင် အလုပ်လုပ်ရာ၌ မည်သည့် နည်းလမ်းများကို လိုက်နာရမည် ဆိုသည်ကိုမူ မပြော ပြရသေးပါ (**နည်းလမ်း ကွဲပြားမှုများစွာ** (https://nvie.com/posts/a-successful-git-branching-model/) ရှိကြသည်)။
- **GitHub**: Git သည် GitHub မဟုတ်ပါ။ GitHub တွင် **pull requests** (https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests) ဟုခေါ်သော အခြား ပရောဂျက်များသို့ Code ပံ့ပိုးပေးသည့် သီးခြား နည်းလမ်း ရှိပါသည်။
- **Other Git providers**: GitHub သည် သီးခြား အထူး မဟုတ်ပါ - **GitLab** (https://about.gitlab.com/) နှင့် **BitBucket** (https://bitbucket.org/) အပါအဝင် Git Repository Hosts များစွာ ရှိကြပါသည်။

Resources

- **Pro Git** (https://git-scm.com/book/en/v2) သည် **အလွန် ဖတ်ရှုသင့်သော စာအုပ်** ဖြစ်ပါသည်။ အခန်း ၁ မှ ၅ ထိ ဖတ်ရှုလိုက်ပါက Data model ကို နားလည်ထားပြီးဖြစ်၍ Git ကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်မည် ဖြစ်သည်။ နောက်ပိုင်း အခန်းများတွင် စိတ်ဝင်စားဖွယ် အဆင့်မြင့် အကြောင်းအရာများ ပါဝင်သည်။
- **Oh Shit, Git!?!** (https://ohshitgit.com/) သည် အတွေ့ရများသော Git အမှားများမှ မည်သို့ ပြန်လည် ကုစားရမည်ကို ရေးသားထားသော လမ်းညွှန် အတို ဖြစ်သည်။
- **Git for Computer Scientists** (https://eagain.net/articles/git-for-computer-scientists/) သည် Git ၏ Data model ကို Pseudocode နည်းပါးစွာဖြင့် ပုံကားချပ်များစွာ အသုံးပြု၍ ရှင်းလင်းထားသော စာတမ်း ဖြစ်သည်။
- **Git from the Bottom Up** (https://jwiegley.github.io/git-from-the-bottom-up/) သည် စိတ်ဝင်စားသူများအတွက် Data model ထက်ကျော်လွန်သော Git ၏ အသေးစိတ် အကောင်အထည်ဖော်မှုများကို ရှင်းလင်းထားခြင်း 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 Git သင်ကြားပေးသော Browser အခြေပြု ဂိမ်း ဖြစ်သည်။

Exercises

1. Git နှင့် ပတ်သက်၍ အတွေ့အကြုံ မရှိသေးပါက **Pro Git** (https://git-scm.com/book/en/v2) ၏ ပထမ အခန်း အနည်းငယ်ကို ဖတ်ပါ သို့မဟုတ် **Learn Git Branching** (https://learngitbranching.js.org/) သင်ခန်းစာကို ပြုလုပ်ပါ။ ပြုလုပ်နေစဉ်အတွင်း Git command များကို Data model နှင့် ယှဉ်တွဲ စဉ်းစားပါ။
2. ဤအတန်း၏ ဝဘ်ဆိုက် repository (https://github.com/missing-semester/missing-semester) ကို Clone လုပ်ပါ။

```
1. သမိုင်းကြောင်းကို Graph အဖြစ် ကြည့်ရှုရှု လေ့လာပါ။
1. `README.md` ကို နောက်ဆုံး ပြင်ဆင်ခဲ့သူမှာ မည်သူနည်း။ (အချက်အလက်- `git log` ကို argument ဖြင့် သုံးပါ)။
1. `_config.yml` ၏ `collections:` စာကြောင်းကို နောက်ဆုံး ပြင်ဆင်ခဲ့သည့် commit ၏ message မှာ ဘာနည်း။ (အချက်အလက်- `git blame` နှင့် `git show` ကို သုံးပါ)။
```

1. Git လေ့လာရာတွင် အတွေ့ရများသော အမှားတစ်ခုမှာ Git ဖြင့် မစီမံသင့်သော ဖိုင်ကြီးများကို Commit လုပ်မိခြင်း သို့မဟုတ် အရေးကြီးသော အချက်အလက်များကို ထည့်သွင်းမိခြင်း ဖြစ်သည်။ Repository သို့ ဖိုင်တစ်ခု ထည့်ပါ၊ Commit လုပ်ပါ၊ ထို့နောက် သမိုင်းကြောင်းထဲမှ ထိုဖိုင်ကို ပယ်ဖျက်ပါ (ဤနေရာတွင် (https://help.github.com/articles/removing-sensitive-data-from-a-repository/) ကြည့်ရှုနိုင်သည်)။
2. GitHub မှ Repository တစ်ခုကို Clone လုပ်ပါ၊ ရှိပြီးသား ဖိုင်တစ်ခုကို ပြင်ပါ။ `git stash` runလိုက်သည့်အခါ မည်သို့ ဖြစ်သွားသနည်း။ `git log --all --oneline` runသည့်အခါ မည်သည်ကို တွေ့ရသနည်း။ `git stash pop` run၍ ပြုလုပ်ခဲ့သည်ကို ပြန်ဖျက်ပါ။ မည်သည့် အခြေအနေတွင် ဤသည်မှာ အသုံးဝင်သနည်း။
3. Command line tool အများစုကဲ့သို့ပင် Git တွင် `~/.gitconfig` ဟုခေါ်သော Configuration file (dotfile) ပါရှိသည်။ `~/.gitconfig` တွင် Alias တစ်ခု ဖန်တီး၍ `git graph` runလိုက်ပါက `git log --all --graph --decorate --oneline` ၏ output ရရှိအောင် ပြုလုပ်ပါ။ `~/.gitconfig` ဖိုင်ကို တိုက်ရိုက် ပြင်ဆင်ခြင်း (https://git-scm.com/docs/git-config#Documentation/git-config.txt-alias) ဖြင့်လည်းကောင်း၊ `git config` command ဖြင့်လည်းကောင်း ပြုလုပ်နိုင်ပါသည်။ Git alias အကြောင်းကို ဒီမှာ (https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases) ကြည့်နိုင်ပါသည်။
4. `git config --global core.excludesfile ~/.gitignore_global` ကို runပြီးနောက် `~/.gitignore_global` တွင် Global ignore pattern များကို သတ်မှတ်နိုင်ပါသည်။ ဤသည်မှာ Git အသုံးပြုမည့် Global ignore file နေရာကို သတ်မှတ်ပေးခြင်း ဖြစ်သော်လည်း ထိုလမ်းကြောင်းတွင် ဖိုင်ကို ကိုယ်တိုင် ဖန်တီးရန် လိုအပ်ဆဲ ဖြစ်သည်။ သင်၏ Global gitignore ဖိုင်တွင် OS သို့မဟုတ် Editor သီးသန့် ယာယီဖိုင်များ (ဥပမာ `.DS_Store`) ကို Ignore လုပ်ရန် သတ်မှတ်ပါ။

5. ဤအတန်း၏ ဝဘ်ဆိုက် repository (https://github.com/missing-semester/missing-semester) ကို Fork လုပ်ပါ။ စာလုံးပေါင်း မှားယွင်းမှု သို့မဟုတ် တိုးတက်ကောင်းမွန်အောင် ပြုလုပ်နိုင်မည့် အရာကို ရှာပြီး GitHub တွင် Pull Request တင်ပါ (ဤနေရာတွင် (https://github.com/firstcontributions/first-contributions) ကြည့်နိုင်ပါသည်)။ အသုံးဝင်သော PR များကိုသာ တင်ပေးပါ (ကျေးဇူးပြု၍ Spam မလုပ်ပါနှင့်!)။ တိုးတက်အောင် ပြုလုပ်စရာ မတွေ့ပါက ဤလေ့ကျင့်ခန်းကို ကျော်နိုင်ပါသည်။

🔗 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 message များ	https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html	
6. ကောင်းမွန်သော commit message များ ရေးသားရခြင်း အကြောင်းအရင်းများ	https://chris.beams.io/posts/git-commit/	
7. စိတ်ကြိုက် ပြင်ဆင်နိုင်စွမ်း မြင့်မားသည်	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. နည်းလမ်း ကွဲပြားမှုများစွာ	https://nvie.com/posts/a-successful-git-branching-model/	
15. pull requests	https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests	

Resource / Description	URL	Micro QR
16. GitLab	https://about.gitlab.com/	
17. BitBucket	https://bitbucket.org/	
18. Oh Shit, Git!?!	https://ohshitgit.com/	
19. Git for Computer Scientists	https://eagain.net/articles/git-for-computer-scientists/	
20. Git from the Bottom Up	https://jwiegley.github.io/git-from-the-bottom-up/	
21. How to explain git in simple words	https://smusamashah.github.io/blog/2017/10/14/explain-git-in-simple-words	
22. Learn Git Branching	https://learngitbranching.js.org/	
23. ဤအတန်း၏ ဝဘ်ဆိုက် repository	https://github.com/missing-semester/missing-semester	
24. ဤနေရာတွင်	https://help.github.com/articles/removing-sensitive-data-from-a-repository/	
25. တိုက်ရိုက် ပြင်ဆင်ခြင်း	https://git-scm.com/docs/git-config#Documentation/git-config.txt-alias	
26. ဒီမှာ	https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases	
27. ဤနေရာတွင်	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

(2020 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/